Fall 1997

# Modeling of deformed swept volumes with SDE and its applications to NC simulation and verification

Feng Lu
*New Jersey Institute of Technology*

# Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

# ABSTRACT

## MODELING OF DEFORMED SWEPT VOLUMES WITH SDE AND ITS APPLICATIONS TO NC SIMULATION AND VERIFICATION

by
Feng Lu

Representation of swept volumes has important applications in NC simulation and verification as well as robot-motion planning. Most research on the representation of swept volumes has been limited to rigid objects. In this study, a sweep deferential equation (SDE) approach is presented for the representation of deformed swept volumes generated by flexible objects.

The deformed swept volume analysis is integrated with machining physics to account for tool deformation/deflection for the NC simulation. End milling is modeled and analyzed and the tool deformations are calculated and integrated with the SDE program. A program is developed in C++ for the generation of deformed swept volumes. Using Boolean subtraction, the deformed swept volume of the tool is cut from the workpiece to simulate the machined part. It is shown that this representation approach constitutes an efficient and accurate NC simulation technique for collision detection, geometric verification as well as surface error prediction.

# MODELING OF DEFORMED SWEPT VOLUMES WITH SDE AND ITS APPLICATIONS TO NC SIMULATION AND VERIFICATION

by
Feng Lu

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Mechanical Engineering

Department of Mechanical Engineering

October 1997

## APPROVAL PAGE

# MODELING OF DEFORMED SWEPT VOLUMES WITH SDE AND ITS APPLICATIONS TO NC SIMULATION AND VERIFICATION

**Feng Lu**

---

Dr. Ming C. Leu, Thesis Advisor — Date
Professor of Department of Mechanical Engineering, NJIT, Newark, NJ

---

Dr. Dennis Blackmore, Committee Member — Date
Professor of Department of Mathematics, NJIT, Newark, NJ

---

Dr. Zhiming Ji, Committee Member — Date
Associate Professor of Department of Mechanical Engineering, NJIT, Newark, NJ

# BIOGRAPHICAL SKETCH

**Author:** Feng Lu

**Degree:** Master of Science

**Date:** October, 1997

**Undergraduate and Graduate Education:**

- Master of Science in Mechanical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 1997

- Bachelor of Science in Precision Instruments,
  Tsinghua University, Beijing, China, 1993

**Major:** Mechanical Engineering

## ACKNOWLEDGEMENT

I would like to express my deepest thanks to my thesis advisor Dr. Ming C. Leu, the sponsored chair in manufacturing productivity, for his resourceful and insightful advice and support during my research at New Jersey Institute of Technology. I would also like to express my special appreciation to Dr. Denis Blackmore and Dr. Zhiming Ji for their advice and serving on my advisory committee.

Many of my fellow graduate students in the Robotics & Intelligent Manufacturing Laboratory at NJIT are also deserving of recognition for their help, especially for Dr. Liping Wang's support and assistance.

## TABLE OF CONTENTS

## TABLE OF CONTENTS
## (Continued)

## LIST OF FIGURES

## TABLE OF FIGURES
## (Continued)

# CHAPTER 1

# INTRODUCTION

This research project consists of three major topic areas:

1. Deformed swept volume analysis and computation
2. Milling process and tool deformation modeling and calculation
3. NC simulation and verification

The objective of this research is to develop an NC simulation module which can generate the deformed swept volumes of tools with deformation in end milling by integrating machining physics with geometric NC simulation & verification. To fulfill this objective, two programs are developed. One is a sweep generator which can compute and represent the deformed swept volume with general spatial deformation. The other is a program to calculate the linear and nonlinear deformations of a tool in end milling process and to integrate these physical deformations with the sweep generator.

In this thesis, the motivation and background of the research are introduced first; the objective and main tasks are discussed second; details of the research follow; results and examples of the implementations are described, and finally, conclusions and some possible future work are suggested.

## 1.1 Motivation

In order to increase productivity in manufacturing, more accurate and faster NC simulation systems are increasingly needed to analyze the performance of the machining process. Currently, a lot of commercial CAD/CAM software packages such as CATIA, I-

DEAS, Pro-Engineer are used popularly in mechanical design and manufacturing. These software packages enable quick changes to design and generation of the resultant NC tool path planning & NC check. However, they fail to consider the machining process physics which can produce errors in the machined part, such as tool deformations, tool wearing, machine system vibration, machine temperature increases, etc. When demands for high speed and accuracy are moderate, as in most of the common machining processes, the errors resulting from the tool deformation and vibration can be overlooked. However, the machining process with high speed and accuracy are used more and more often. In these situations, the tool deformation is one of the most significant error factors.

There have been many studies on the cutting force modeling (Devor and Sutherland, 1982, 1986, 1987; Altintas, et al. 1991, 1993, 1995) and end mill tool deformations (Kline and Devor, 1982; Takata, et al., 1989; Armarego, et al. 1990, 1991, 1992). However, few of the studies focused on integrating the predicted cutting force model and tool deformation with geometric (visual) NC simulation and verification. Swept volumes, as a subclass of configurations in the area of solid modeling, have important applications in manufacturing design and practice, such as NC simulation and verification, computational geometry design, robot motion planning, etc. The sweep differential equation and boundary-flow method have been analyzed and used to represent the swept volumes (Blackmore, Leu, et al. 1991, 1992) and implemented successfully in NC simulation and verification (Leu, Blackmore, et al., 1992,1995,1996) as well as robot-motion planning (Deng, Leu, 1996). Also, some research work has been done on the theoretical extension of SDE to include the deformation for representing the deformed swept volumes (Blackmore, Leu, 1994). Tool deformation, which includes linear

deflection and nonlinear deformation, can be integrated into the swept volumes of tool in end milling. In this way, the deformed swept volumes of the milling cutter can be generated and subtracted from the workpieces to simulate the NC machining process more accurately.

## 1.2 Objective and Main Tasks

The aim of this project is to develop an NC simulation and verification module using the swept volume representation approach. One of the objectives is to extend the existing theory and algorithm of SDE to include the spatial deformation. The physical deformation, specifically the tool deformation/deflection in NC machining, is analyzed and calculated. Then by integrating the physical deformation with SDE method the deformed swept volumes of the tool in machining process are generated. Using the Boolean subtraction, the swept volume of the tool are cut out from the workpiece to simulate the machined part. The surface error of the simulated machined part is predicted. The area/surface patch where the error exceeds the tolerance is also predicted.

The major tasks of this project are:

1) Survey and analysis of the milling process models, including cutting force models, system dynamics models, tool deformation models.
2) Development of a program to generate the linear and nonlinear deformations of the end mill.
3) Representation and computation of the deformed swept volumes by the SDE method. A sweep generator, which can represent the deformed swept volumes, is developed.

4) Integration of the physical deformation of endmill into the sweep generator to represent the swept volumes of end mill under deformation.

5) Analysis and implementation in NC simulation and verification.

# CHAPTER 2

# ANALYSIS AND REPRESENTATION OF DEFORMED SWEPT VOLUMES BY SDE

## 2.1 Related Studies

The concept of swept volume was initiated in the late 1970's and 1980's to study manufacturing automation strategies. Many of the applications of swept volume studies have been proposed and implemented for the simulation of the material removal process in machining, detection of machining collision as well as vehicle motion planning. A great amount of effort has been devoted to developing fast and accurate methods to represent the swept volumes. The most commonly used methods are envelope theory, z-buffer, ray-casting methods and sweep differential equations. (Wang & Wang , 1986; Weld et al. 1990; Leu et al. , 1986; Narvekar, 1991; Sambandon, 1988, 1990, et al.)

The envelope technique (Wang & Wang , 1986) was one of the earliest attempts to compute the swept volumes generated by developable surfaces. Based on the theory, Sambandan (1988) developed a 5-axis NC simulator for flat-end, ball-end and fillet-end cutters by deriving the parametric representation of the boundary surface of the cutter swept volumes. Narvekar (1991) used envelope equations to derive the swept volumes of general 7-parameter APT tools and also conducted intersection calculation. However, the fact that swept volume may be formed with some self-intersected envelope surfaces and the envelope method is essentially local in nature makes the envelope method somewhat deficient. Some other methods were also used by researchers to represent the swept volumes, such as the ray_casting engine (Menon & Robinson, 1993).

Blackmore and Leu (Blackmore & Leu, 1990, 1992) introduced a new approach, the sweep differential equation method, for the study of swept volumes. This approach fully exploits the Lie group structure of the set of Euclidean motion and thereby enables the problem of swept volume to be reformulated as the problem of solving deferential equations. In the recent years of research conducted by them, the potential of this approach for automated manufacturing applications, robot motion planning have been discussed (Blackmore et al. 1992; Deng et al., 1994, 1996).

The SDE theory was implemented initially for two dimensional objects under planar motion and a computer program for the representation of 2D swept volumes was developed by Jiang (1993). Qin et al. (1994) modified Jiang's work by introducing a combination of envelope differential equation with the sweep deferential equation method to generate the grazing points set of the swept volumes boundary more efficiently. Wang et al. (1995, 1996) used the SEDE (sweep-envelop differential equation) method to represent the swept volumes generated by a 7-parameter APT tool under general motion in NC machining and implemented it in 5-axis NC machining simulation and verification.

The sweep differential equation approach and the boundary flow method can be extended to include the deformation of an object under general motion. The research on the analysis and modeling of the deformed swept volumes was conducted by Blackmore and Leu (1994) and examples on 2D objects with deformation under planar motion were also given. However, a computer program for 3D objects with deformation under general motion have not been developed. Also, the deformation (linear or general) was discussed theoretically but the physical deformation has not been discussed and implemented yet. As an application in NC simulation and verification, the SDE method can be extended to

include the physical deformation of the tool. By integrating the calculated (predicted) tool deformation in the SDE method, we can create a more accurate NC verification module which can not only allow the user to check the material removal process and collision, but also let the user check the accuracy of the simulated machined part.

## 2.2 Sweep Differential Equation with General Deformation

### 2.2.1 Preliminaries of SDE

A Euclidean n space is denoted as $R^n$, which R is the field of real numbers. The space consists of all n tuples $x=\{(x_1, x_2, \ldots\ldots x_{n-1}, x_n) : x_1, x_2, \ldots\ldots x_{n-1}, x_n \in R\}$, and supports the standard inner product of any two $A, B \in R^n$, $A=\{a_1, a_2, \ldots\ldots a_{n-1}, a_n\}$, $B=\{b_1, b_2, \ldots\ldots b_{n-1}, b_n\}$

$$< A, B > = \sum_{k=1}^{n} a_k b_k \tag{2.1}$$

The object M to be swept, occupying 3 dimensional space $R^3$, is assumed to be closed and bounded with a boundary surface $\partial M$ which is piecewise smooth. In practical terms, the object considered, such as cutting tool, robot, manipulator arms, etc., has a smooth boundary except for a finite number of edges and vertices.

A sweep is a family of rigid motions which comprise rotation and translation. More precisely, a smooth sweep $\sigma$ in $R^n$ is a smooth mapping:

$$\sigma:[0,1] \rightarrow E(n) \tag{2.2}$$

where E(n) is an analytical Lie group of Euclidean motion in $R^n$ such that $\sigma$ at t=0, denoted by $\sigma_0$ , is the identity mapping id. From the definition, if we let $\sigma_t$ represent a sweep at time t, it can be written as

$$\sigma_t(x_0) = X(t) = \xi(t) + A(t)x_0 \tag{2.3}$$

where: $\zeta$: [0,1]→$R^n$ and A: [0,1]→SO(n) are smooth functions representing the translation and rotation motion of the sweep.

If M is a piecewise smooth object in $R^n$ with a smooth sweep, the set

$$\sigma_t(M) = M(t) = \{\ \sigma_t(x);\ x \in M\ \} \tag{2.4}$$

is called the t section of M under sweep $\sigma$. The swept volume of M generated by $\sigma$ is

$$S_\sigma(M) = \{\ \sigma_t(M) : 0 \le t \le 1\ \} \tag{2.5}$$

Solving equation 2.3 for $x_0$, we have

$$x_0 = A^{-1}(t)(x - \xi(t)) \tag{2.6}$$

Differentiating equation 2.3 with respect to time t and substituting $x_0$ with equation 2.6, we can derive the sweep vector field (SVF) of a smooth sweep

$$X_\sigma = \dot{x}(t) = \dot{\xi}(t) + \dot{A}(t)A^T(t)(x - \xi(t)) \tag{2.7}$$

### 2.2.2 Boundary-Flow Formula

In the boundary flow method (BFM), the sweep vector field (SVF) partitions the boundary of the t section of M into ingress, egress and grazing points which are defined as following.

**Definition**: Let M be a piecewise smooth object, the tangency function for a sweep of the object M is defined as

$$T(x,t) = < X_\sigma(x,t)\ ,\ N(x,t) > \tag{2.8}$$

where <a, b> denotes the inner product of a, b in $R^n$ , N(x,t) is the unit outward normal vector on the smooth part of $\partial$M at point P(x);

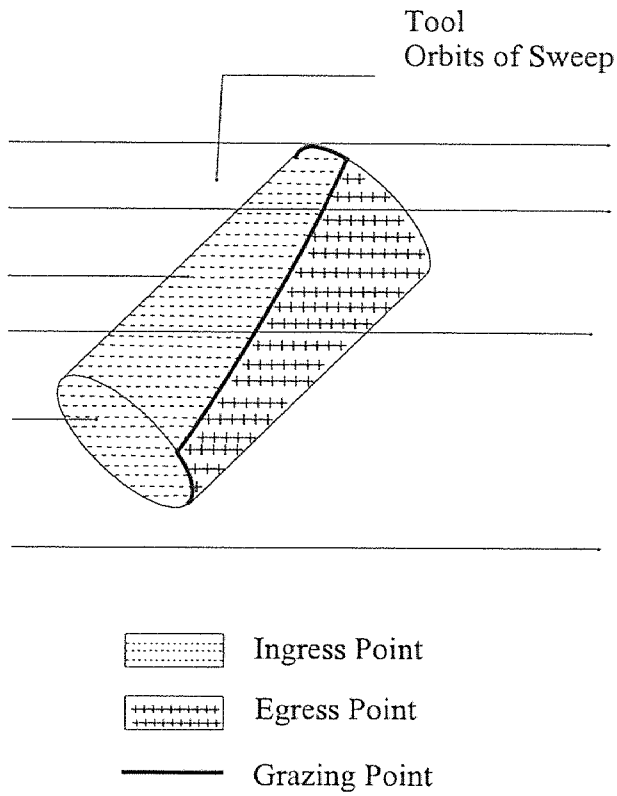



**Figure 2.1** Object boundary partition

**Definition**: the set of ingress (egress) points of M(t), denoted by $\partial\_M(t)$ ($\partial\_M(t)$), consists of all points $x \in \partial M(t)$ at which $X_\sigma(x, t)$ points into (out of) the interior of M. Those points that are neither ingress nor egress points are called grazing points which are denoted by $\partial_0 M(t)$, as shown in Fig. 2.1.

Also, we can define them in the context of tangency function:

$$\begin{aligned} \partial\_M(t) &= U\{T(x,t) < 0,\ x \in M,\ t \in [0,1]\} \\ \partial_+ M(t) &= U\{T(x,t) > 0,\ x \in M,\ t \in [0,1]\} \\ \partial_0 M(t) &= U\{T(x,t) = 0,\ x \in M,\ t \in [0,1]\} \end{aligned} \tag{2.9}$$

Let the object M and sweep σ be defined as above, the boundary of the swept volume is given by

$$\partial S(M) = G(M) \setminus T(M) \tag{2.10}$$

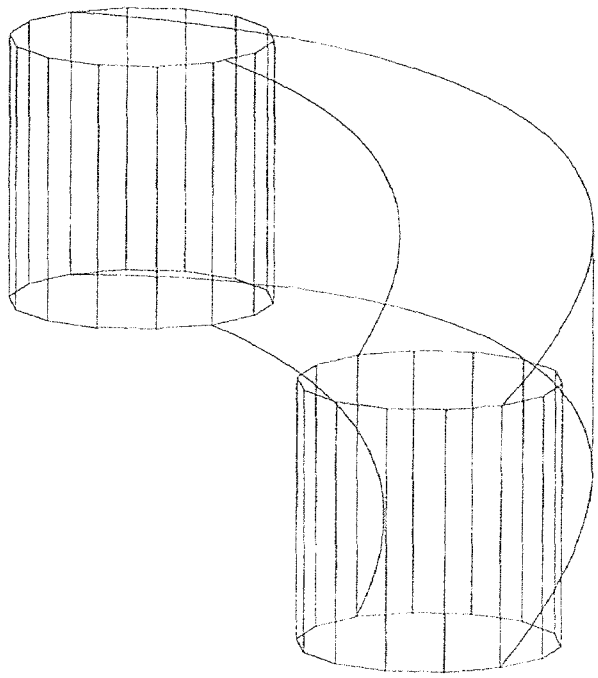


**Figure 2.2** A typical swept volume in 3D space

where G(M) = $\partial\_M(0)$ U $\partial_+M(1)$ U $\{\partial_0M(t):0<t<1\}$ is the candidate boundary set which consists of the ingress points of object M at t=0, egress points of M at t=1 and all the grazing points in between. T(M) denotes the trimming set (or, the intersecting set) which belongs to the interior of some t section of M and thus does not belong to the portion of the boundary of swept volume. A typical swept volume of an object in 3D space is shown in figure 2.2.

### 2.2.3 Swept Volume with Deformations

The sweep differential equation and boundary flow method can be extended to include objects experiencing deformations. Although there are several special cases of deformation which are easier to be implemented in SDE, we examine the case with general deformation. In the following section, we will analyze and discuss the deformed swept volumes with general spatial deformation.

Given a piecewise smooth object under general motion and deformation, as we can derive from the non-deformation swept equation, a smooth sweep with general deformation can be written as:

$$\sigma_t(x_0) = X(t) = \xi(t) + A(t)(L(t) * x_0 + D_n(x_0, t)) \tag{2.11}$$

where $L(t)\cdot x_0$ and $D_n(x_0, t)$ denote the linear and nonlinear deformation, respectively. We can rewrite equation 2.11 in a more concise form:

$$\sigma_t(x_0) = X(t) = \xi(t) + B(t)x_0 + D(x_0, t) \tag{2.12}$$

where $B:[0,1]\rightarrow GL^+(n)$ and $H: R^n\times[0,1]\rightarrow R^n$ are smooth mappings such that $A(0)=I$, $D(x_0,0) = \partial_t D(x_0,0) = 0$ for all points on M. Differentiating equation 2.12 with respect to t, we can derive

$$X_\sigma = \dot{x}(t) = \dot{\xi}(t) + \dot{B}(t)x_0 + \partial_t D(x_0, t) \tag{2.13}$$

By solving equation 2.13 for $x_0$, we get

$$x_0 = G^{-1}(x - \xi(t)) = B^{-1}(t)(x - \xi(t)) + P(x - \xi(t), t) \tag{2.14}$$

where

$$P(x,t) = B^{-1}(t)D(B^{-1}(t)x + P(x,t),\ t)$$

Combining equation 2.13 and equation 2.14, we obtain the SDE for general deformation:

$$\begin{aligned} X_\sigma &= \dot{x}(t) = \dot{\xi}(t) + W(x,t) \\ &= \dot{\xi}(t) + \dot{B}(t)B^{-1}(t)(x-\xi(t)) + \partial_t D(B^{-1}(t)(x\ -\xi(t)) + P(x-\xi(t)),t) \end{aligned} \tag{2.15}$$

As we discussed in the preliminary section, we use the boundary flow formula to represent the boundary of the swept volume. The tangency function, which is used to identify the ingress/egress and grazing points, can be generated from the sweep of the initial outward normal vector $N_0$ of the smooth part of the object M. Given a piecewise smooth object M, with outward normal vector N and a vector Y tangent to $\sigma_t(M)$ at $\sigma_t(x_0)$, we note that

$$(B(t)+\partial_x D(x,t))^{-1}Y = \partial_x \sigma_t^{-1}(Y) \tag{2.16}$$

is tangent to the interior of object M at x. Therefore,

$$< (B(t)+\partial_x D(x,t))^{-1}Y, N_0 > \ = 0$$

According to the properties of the inner product of the vector,

$$< (B(t)+\partial_x D(x,t))^{-1}Y, N_0 > \ = \ < Y, [(B(t)+\partial_x D(x,t))^{-1}]^T N_0 > \ = 0$$

In consequence, $[(B(t)+\partial_x D(x,t))^{-1}]^T N_0$ is orthogonal to vector Y and thus is the outward normal vector field. The tangency function, therefore, can be expressed as

$$T(x,t) =< X_\sigma(x,t)\ ,\ N(x,t) >=< X_\sigma(x,t), [(B(t)+\partial_x D(x,t))^{-1}]^T N_0 > \tag{2.17}$$

where $\partial_x D(x,t)$ stands for, in 3 dimensional space:

$$\partial_x D(x,t) = \begin{bmatrix} \frac{\partial D_x}{\partial x} & \frac{\partial D_x}{\partial y} & \frac{\partial D_x}{\partial z} \\ \frac{\partial D_y}{\partial x} & \frac{\partial D_y}{\partial y} & \frac{\partial D_y}{\partial z} \\ \frac{\partial D_z}{\partial x} & \frac{\partial D_z}{\partial y} & \frac{\partial D_z}{\partial z} \end{bmatrix} \tag{2.18}$$

In the practical point of view, most of the boundaries of the objects such as machining tool or robot arms can be approximated as analytical surfaces. In this context, another relative simple method for calculating the outward normal vector field is used. Given a surface in 3 dimensional space which is piecewise smooth and can be expressed as a parametric equation:

$$\begin{cases} x = x(u,v) \\ y = y(u,v) \\ z = z(u,v) \end{cases}$$

Let $\vec{r}_m$ stand for the vector of any point m on this surface, $\vec{r}_m(t) = x(u,v)\vec{i} + y(u,v)\vec{j} + z(u,v)\vec{k}$ , the outward normal vector of the surface at point m, as shown in figure 2.3, is :

$$\vec{N}_m(t) = \frac{\partial\, \vec{r}_m(t)}{\partial u} \times \frac{\partial\, \vec{r}_m(t)}{\partial v} \tag{2.19}$$

The tangency function, therefore, can be described as

$$\begin{aligned} T(x,t) &=< X_\sigma(x,t), N(x,t) > \\ &= X_\sigma(x,t)(y_u z_v - y_v z_u) \\ &\quad + X_\sigma(y,t)(x_v z_u - x_u z_v) + X_\sigma(z,t)(x_u y_v - x_v y_u) \end{aligned} \tag{2.20}$$

where $x_u, x_v, y_u, y_v, z_u, z_v$ stand for:

$$x_u = \frac{\partial x(u,v,t)}{\partial u} \quad , \quad x_v = \frac{\partial x(u,v,t)}{\partial v} \quad \cdots$$

and $X_\sigma(x,t), X_\sigma(y,t), X_\sigma(z,t)$ are the components of $X_\sigma(x,t)$ in x, y, z directions.

And the partial derivatives of $x(u,v,t)$ with u, v are

$$\frac{\partial x(u,v,t)}{\partial u} = B(t)\cdot\frac{\partial x_0(u,v)}{\partial u} + \frac{\partial D(x_0(u,v),t)}{\partial u}$$
$$\frac{\partial x(u,v,t)}{\partial v} = B(t)\cdot\frac{\partial x_0(u,v)}{\partial v} + \frac{\partial D(x_0(u,v),t)}{\partial v} \qquad (2.21)$$

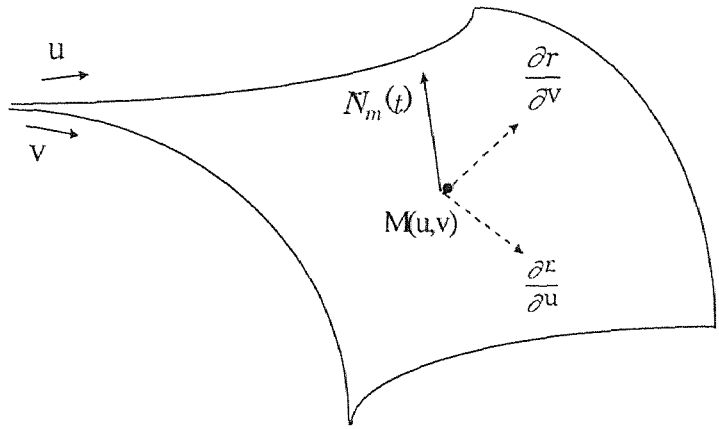



**Figure 2.3** Normal Vector of an Analytical Surface

Let object $M$ and sweep $\sigma$ be as described above, the boundary of the deformed swept volume is given by the formula

$$\partial S_\sigma(M) = C_\sigma(M) \setminus T_\sigma(M) \qquad (2.22)$$

where:

$C_\sigma(M) := \partial_- M(0) \cup \partial_+ M(1) \cup G_\sigma(M)$, is called the *candidate set*,

$G_\sigma(M) := \cup\{\partial_0 M(t) : 0 \le t \le 1\}$ is the *grazing set* of the swept volume and

$T_\sigma(M)$, called the *trim set*, consists of those points of $C_\sigma(M)$ that belong to the interior of $S_\sigma(M)$.

# CHAPTER 3

# ANALYSIS AND MODELING OF END MILLING PROCESS

## 3.1 Overview

One of the most common metal removal operations used in industry is end milling. In order to improve the quality and productivity, accurate models of the milling process are required for both analysis and prediction of the quality of the machining process. Such analysis and prediction have potentials, for example, to greatly reduce the time required for NC verification in test cuts and improve the quality of the finished surface of the product. The milling process model, which includes the cutting force model, flexible tool deformation model and sometimes the dynamics (chattering) model, can be used to predict the cutting force, tool breakage, tool wearing, chattering condition as well as surface error.

In the past several years, much research work has been conducted on the milling process modeling. Several types of cutting force models and tool deformation/deflection model have been presented and discussed. According to the sophistication and accuracy, the models can be classified as:

I) Average rigid force model;

II) Distributed rigid force model;

III) Distributed force with flexible tool deflection feedback;

IV) Distributed force with system dynamics.

The first two models, relying on the relationship between metal removal rate (MRR) and cutting force, do not take into account the effect of tool deflection on the

cutting force and therefore have some deficiencies. Yet they are still very popular models which are widely used in cutting force approximation and prediction. (Wang, 1988; DeVor, and Kline 1980, 1985; Tlusty, 1985). The third one does consider the effects of the tool deflection on the uncut chip thickness and uses tool deflection as a feedback which affects the cutting force. This model is more complicated and accurate than the former ones and has been used by many researchers ( DeVor and Sutherland, 1986; Tlusty 1983; Armarego, 1990,1991; Meng and Feng, 1996, et.). The fourth one, which is the most complicated model, considers the system dynamics between tool and workpiece. The dynamics is assumed to be second order damped vibration system as mentioned by Smith and Tlusty (1991) and used by Altintas ( 1995).

Another topic in the end milling modeling and analysis relates to the tool deflection and surface error. Kline and DeVor et al. (1982) presented a flexible tool model which modeled the end mill as a cantilever beam rigidly supported by the holder. Takata, Tsai, (1989) used another model in their cutting simulation system, by only considering the tool deflection with linear displacement and angular displacement between the tool and the chuck. For a complete analysis and modeling of the tool deformation, both linear and nonlinear components of deflection must be considered and combined. Each of these models is used in cutting force prediction which considers the flexible tool feedback. Armarego and Deshpande (1990, 1991, 1992) published three papers in a series, discussing each of the models and their effects on cutting force prediction.

In the following, we will present all the four types of cutting force models and discuss their advantages and deficiencies in application. In order to illustrate the cutting force models more clearly, the tool deflection/deformation model will be discussed first.

## 3.2 Tool Deformation/Deflection Models

In our modeling, both the linear and nonlinear deformations of the tool are modeled and used. We assume that the cutting force is applied at tool tip. If the cutting force is modeled as distributed forces, as in cutting force models II, III and IV, the accumulated cutting force and accumulated moment also can be calculated (for details see cutting force model II, section 3.3.2).

### 3.2.1 Linear Deflection

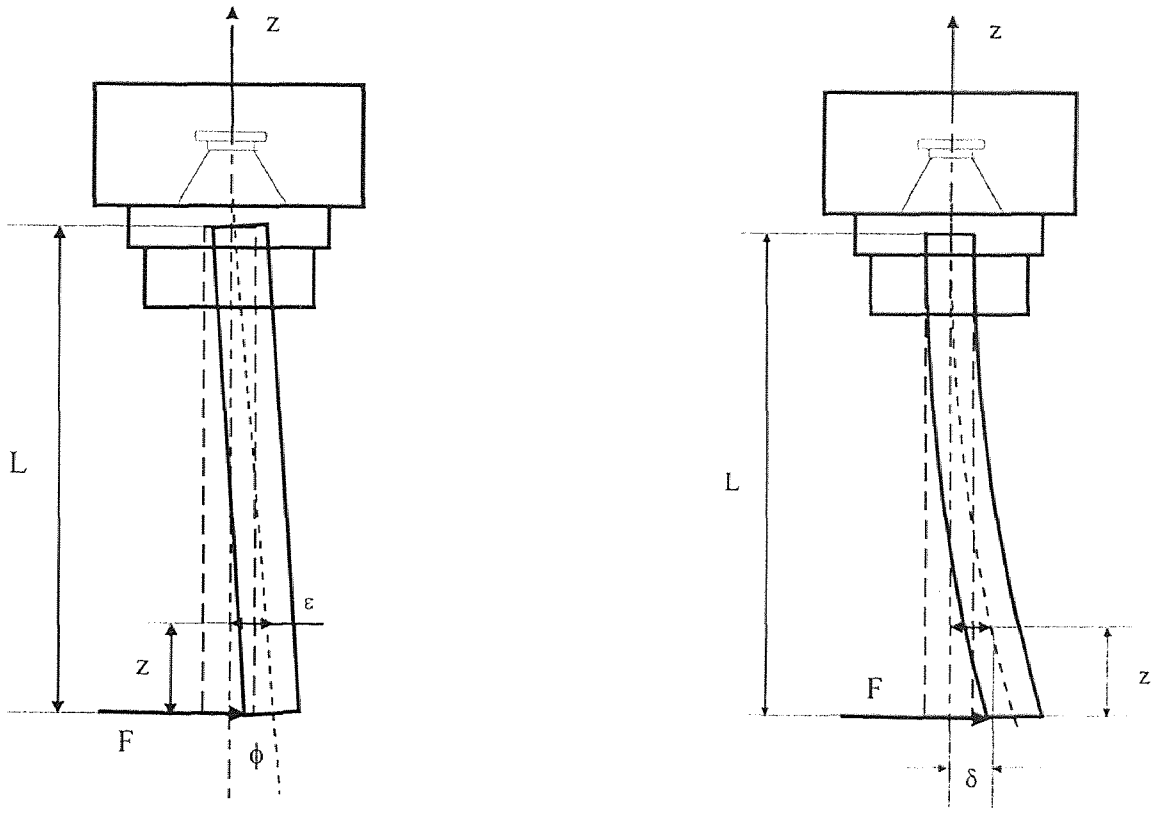
The measurements of tool deflection showed that the interfaces between tool and chuck are the weakest part and the displacement at these parts contribute to most of the tool deflection. As shown in figure 3.1, the linear deflection consists of linear displacement of the tool measurement at the center of the chuck, ε, and the angular displacement of the tool, ϕ.

The linear deflection at each point ( $z$ )of the tool can then be calculated as:

$$\begin{aligned} \varepsilon_x(z) &= E_h * F_x(t) + E_r * M_x(t) * (L - z) \\ \varepsilon_y(z) &= E_h * F_y(t) + E_r * M_y(t) * (L - z) \end{aligned} \tag{3.1}$$

where $F_x$, $F_y$ are the predicted cutting force components in x, y directions. $E_r$, $E_h$ are deflection constants, which can be obtained through experiments by applying some static forces on the tool and measuring the deflections. $M_x$, $M_y$ are the moments produced by the cutting force on the tool top:

$$\begin{aligned} M_x(t) &= F_x(t)L \\ M_y(t) &= F_y(t)L \end{aligned} \tag{3.2}$$




Linear Deflection Model Nonlinear Deformation Model

**Figure 3.1** Tool Deflection/Deformation Model

### 3.2.2 Nonlinear Deformation

For the prediction of nonlinear deformation, a relatively simple but efficient model is adapted by modeling the end mill as a cantilever beam, as shown in figure 3.1. For the cutting force acted on the tip of cutter, the deformation of the tool along the z axis can be calculated as:

$$\begin{aligned} \delta_x(z) &= -F_x(t) \cdot (L-z)^2 \cdot (2L+z) / 6EJ \\ \delta_y(z) &= -F_y(t) \cdot (L-z)^2 \cdot (2L+z) / 6EJ \end{aligned} \tag{3.3}$$

Given the linear deflection and nonlinear deformation calculated as above, the total tool displacement at any point along the z axis can be summed as:

$$D_x(z) = \varepsilon_x(z) + \delta_x(z)$$
$$D_y(z) = \varepsilon_y(z) + \delta_y(z) \tag{3.4}$$

## 3.3 Cutting Force Models

The popular cutting force models for end milling process will be discussed according to the model's sophistication and accuracy.

### 3.3.1 Average Rigid Cutting Force Model

*Cutting force magnitude*:

As one of the most basic, yet still very popular models, the average rigid force model relates the material removal rate (MRR) linearly to the average cutting force. The tool deformation is not considered as a feedback factor that affects chip thickness. According to Smith & Tlusty (1992), the tangential average cutting force $F_t$ can be expressed as:

$$F_t = P_{sp}(MRR) / \upsilon \tag{3.5}$$

where $F_t$: average tangential cutting force

$P_{sp}$: specific power

$MRR$: material removal rate

$\upsilon$: peripheral cutting speed, $\upsilon = \pi dn$

The values of $P_{sp}$ are available in the handbook for different tools, workpieces and machines. Material removal rate (MRR), in general, is given by

$$MRR = Amfn \tag{3.6}$$

where $A$: the cross section area of the uncut chip (For details of calculating A see section 3.4.2

$m$: number of teeth of the cutter

$f$: chip load (feed per tooth)

$n$: spindle speed

The cutting force acting on the normal direction to the cut, $F_s$, is taken as:

$$F_s = F_t / 2 \tag{3.7}$$

The average cutting force is assumed to be acting on the tip of the cutter and the tool deflection/deformation will then be calculated according to this assumption by using equations 3.1 and 3.3.

*Cutting force direction:*

Since equations 3.5 and 3.7 only indicate the magnitude of the average cutting force, we need to identify the direction of the force.

There are two types of milling in general: up milling vs. down milling. As shown in figure 3.2, a difference between these two cutting types is the cutter rotation direction. Figure 3.2 show a cross section of the tool in milling process. For up milling, we can see the cutter tip has two cutting force loading: one is normal force $dF_n$ on the rake face and another is friction force $dF_f$, also on the rake face. Usually, $dF_n$ is larger than $dF_f$. Therefore, the y component of $dF_n$ is also larger than of $dF_f$. That means, the cutting force is in the positive y direction. For down milling, all the y components of $dF_n$ and $dF_f$ are in the negative y direction. That is to say, for up milling, the cutting force in most

times is directed into workpiece and for down milling it directs out from workpiece

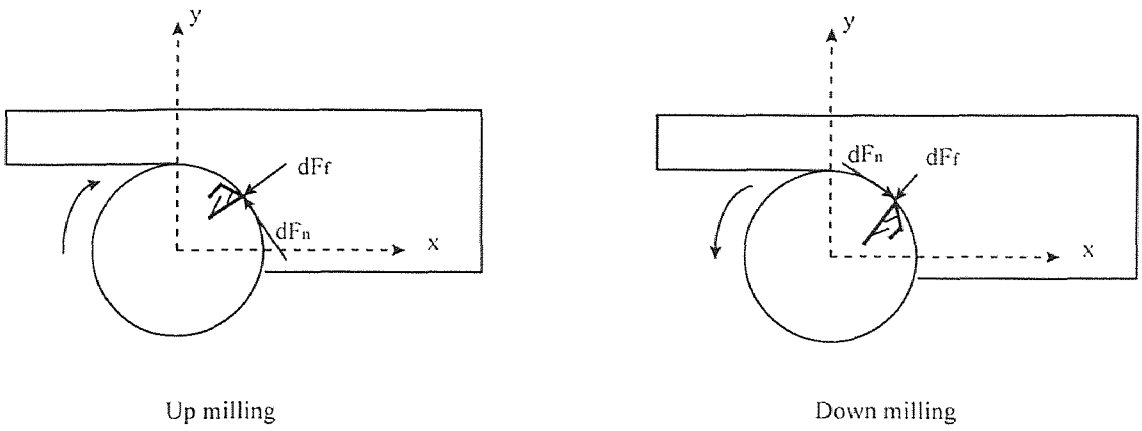



Figure 3.2 Cutting Force Direction

For slot cuts, we can use the same method to identify the cutting force direction.

Advantage: This is a very popular model which is widely used as approximation of average cutting force. It is easy to be used and calculated, and is suitable for implementation with our SDE algorithm.

Disadvantage: It does not consider the cutter geometry and the details of machining in each rotation. Therefore, it is not so accurate.

### 3.3.2 Distributed Rigid Cutting Force Model

In this model, for a more accurate prediction of instantaneous cutting force, the end mill is divided into a series of slices along the tool axis and the milling process is examined angle by angle, flute by flute as shown in figure 3.3. Again, tool deformation is not used as a feedback.

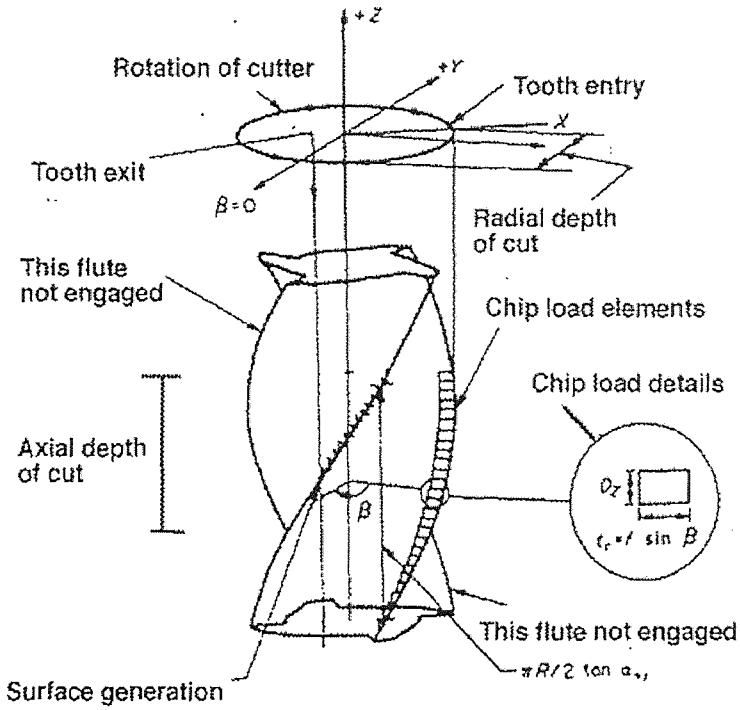



**Figure 3.3** Tool geometry modeling

Several researchers in the past have attempted to develop the chip-force relations for the end milling process. Given a tool with length L divided into K slices, the governing relation between the cutting force and uncut chip thickness for at $k$ th slice, $j$ th flute at orientation angle $\theta_i$, can be given as:

$$\begin{aligned} \Delta F_t(\theta_i, j, k) &= K_t \cdot \Delta z \cdot h(\theta_i, j, k) \\ \Delta F_r(\theta_i, j, k) &= K_r \cdot \Delta z \cdot h(\theta_i, j, k) \end{aligned} \tag{3.8}$$

where $\Delta F_t$ and $\Delta F_r$ are tangential and radial components of the elemental cutting force, respectively;

$K_t$, $K_r$ stand for the coefficients which are obtained by conducting cutting test experiments;

$\Delta z$ and $h$ represent each slice thickness along tool axis and uncut chip thickness, respectively.

The chip thickness of a specified flute at particular slice and angular position depends on several factors such as feedrate, runout and cutting system deformations. The uncut chip thickness is the smallest radial distance between the path the current edge is generating and the machined surface left by the previous m flutes. The chip thickness is thus given by (as shown in Fig. 3.4):

$$h(\theta_i, j, k) = mf_t \sin\theta(i, j, k) + (R_i - R_{i-m}) \tag{3.9}$$

where $f_t$ is the feedrate and $R_i$ stands for the real cutter radius at time instance $i$ with runout. $R_i$ equals to nominal radius $R$ if runout is not considered.

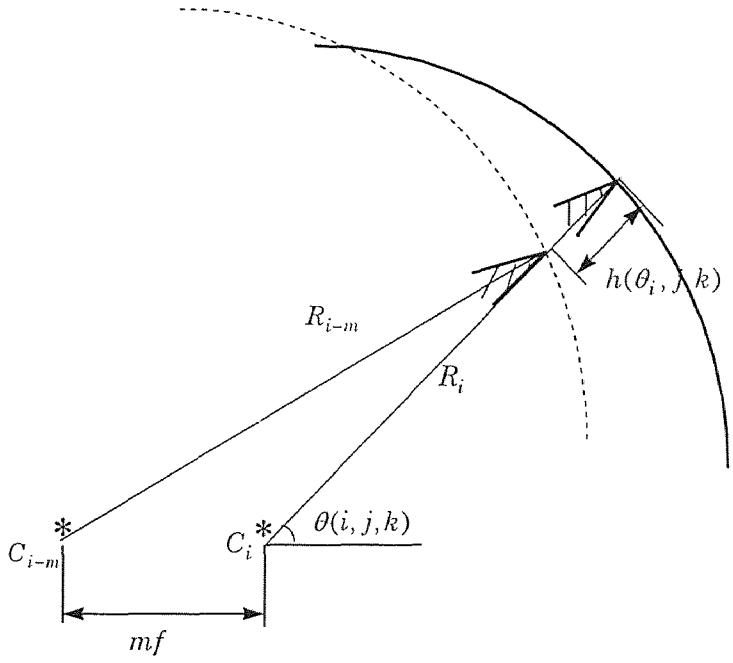



**Figure 3.4** Uncut chip thickness calculation

$\theta(i,j,k)$ is the orientation angle of the cutter edge at time i, flute j and slice k.

$$\theta(i,j,k) = [-\theta_i + 2\pi(j-1)/N_f] + \psi \tag{3.10}$$

where $\theta_i$ is the rotational angle at time i; $N_f$ is the number of flutes and $\psi$ stands for helix angle of the cutting edge.

By substituting equation 3.9 into equation 3.8, cutting force $\Delta F_t(\theta_i,j,k)$, $\Delta F_r(\theta_i,j,k)$ at any slice, any flute and any time instance can be calculated. In order to sum the force for each flute, we can project the cutting force onto $x,y$ direction in general coordinates.

$$\begin{bmatrix}\Delta F_x(\theta_i,j,k)\\ \Delta F_y(\theta_i,j,k)\end{bmatrix} = \begin{bmatrix}-\sin[\theta(i,j,k)] & -\cos[\theta(i,j,k)]\\ \cos[\theta(i,j,k)] & \sin[\theta(i,j,k)]\end{bmatrix}\begin{bmatrix}\Delta F_t(\theta_i,j,k)\\ \Delta F_r(\theta_i,j,k)\end{bmatrix} \tag{3.11}$$

Therefore, the cutting force at each slice and time i is:

$$\begin{aligned}\Delta F_x(\theta_i,k) &= \sum_{j=1}^{N_f} \Delta F_x(\theta_i,j,k)\\ \Delta F_y(\theta_i,k) &= \sum_{j=1}^{N_f} \Delta F_y(\theta_i,j,k)\end{aligned} \tag{3.12}$$

Finally, the accumulated cutting force and its loading position along z axis $l_x, l_y$ is:

$$\begin{aligned}F_x(\theta_i) &= \sum_{k=0}^{K} \Delta F_x(\theta_i,k)\\ F_y(\theta_i) &= \sum_{k=0}^{K} \Delta F_y(\theta_i,k)\end{aligned} \quad \text{and} \quad \begin{aligned}l_x &= M_x(\theta_i)/F_x(\theta_i)\\ l_y &= M_y(\theta_i)/F_y(\theta_i)\end{aligned} \tag{3.13}$$

where $M(\theta_i)$ is the accumulation moments at tool top.

$$\begin{aligned}M_x(\theta_i) &= \sum_{k=0}^{K} \Delta F_x(\theta_i,k)\cdot(L - L\cdot k/K)\\ M_y(\theta_i) &= \sum_{k=0}^{K} \Delta F_y(\theta_i,k)\cdot(L - L\cdot k/K)\end{aligned} \tag{3.14}$$

The tool deflection/deformation will then be calculated according to the accumulated cutting force. Since the accumulated cutting force is not applied at the tool tip, equations 3.1 and 3.3 need a little bit modification:

$$\begin{aligned} M_x(t) &= M_x(\theta_i) \\ M_y(t) &= M_y(\theta_i) \end{aligned} \tag{3.15}$$

and

$$\begin{aligned} \delta_x(z) &= -F_x(t)\cdot(L-z)^2\cdot(2L-3l_x+z)/6EJ \\ \delta_y(z) &= -F_y(t)\cdot(L-z)^2\cdot(2L-3l_y+z)/6EJ \end{aligned} \tag{3.16}$$

where $F_x(t)$ and $F_y(t)$ are the accumulated cutting forces according to equation 3.13.

$l_x$, $l_y$ represent the position along tool axis where accumulated cutting forces was loaded.

Advantage: The cutting force is modeled as distributed and each component is calculated according to different rotational angle, different slice and flute. Therefore, it is more accurate than model I, which only calculates the averaging cutting force.

Disadvantage: It still does not consider the tool deflection/deformation as a feedback on chip thickness and cutting force prediction.

### 3.3.3 Distributed Force with Flexible Tool Deflection Feedback

When we look closer at the machining process, especially at a process with a slim tool or heavy machine load, the tool deflection/deformation is so large that we can not neglect its effect on the cutting force prediction. That's to say, the tool deflection/deformation will

affect the chip thickness calculation. According to the present modeling scheme (DeVor, 1986; Menq 1996), the updated chip thickness model is presented as follows:

The deformation based chip thickness is given by (as shown in Fig. 3.5):

$$\begin{aligned} h(\theta_i, j, k) = {} & mf_t \sin\theta(i, j, k) + (R_i - R_{i-m}) \\ & + \gamma_{x_{i-m}} \cdot \sin\theta(i, j, k) \\ & + \gamma_{y_{i-m}} \cdot \cos\theta(i, j, k) \end{aligned} \tag{3.17}$$

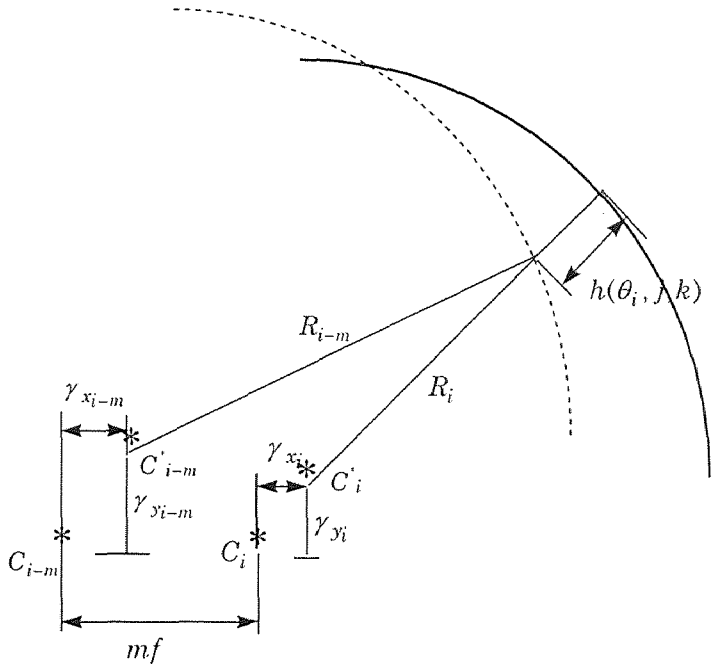



**Figure 3.5** Deformation based chip thickness

where $f_t$ is the feedrate and $R_i$ stands for the real cutter radius at time instance $i$ with runout. $R_i$ equals to nominal radius $R$ if runout is not considered. $\gamma_i$ indicates the total tool deformation at $k$th slice and time instance $i$ .

The cutting force governing equation and tool deflection/deformation are the same as model II discussed in section 3.2.

The basic procedure for the cutting force prediction for this flexible tool based model is:

a) Input tool geometric parameters and machining parameter such as spindle speed, feedrate, as well as material parameters

b) Tool is divided into K slices; Rotational step is set: for example $\Delta\theta = 5^\circ$; $m = 1$

c) During the first rotation, tool is assumed to be rigid. No tool deformation is considered: $\gamma(0) = 0$

(After the first rotation, the tool deformation at the previous time instance is used as the chip thickness update for the current time instance)

d) Calculate the distributed cutting force at each time instance $i$ (rotational angle) according to equations 3.8 and 3.11.

e) Calculate the accumulated cutting force and tool deflection/deformation $\gamma(i)$ according to equations 3.12, 3.13 & 3.1, 15,16

f) Set $i = i+1$, go to the next time instance; go back to step d) and so on.

Advantage: The model considers tool deflection/deformation as a feedback on chip thickness calculation. Therefore, it is more accurate and complex than first two models.

Disadvantage: Computational complexity is increased; system dynamics is still not considered

### 3.3.4 Distributed Force with System Dynamics

The most advanced model, which takes into account the effects of system dynamics between tool system and workpiece, has been developed, Smith and Tlusty (1991). This model was intended to study the dynamics aspects of the machining process. According to their modeling, the system dynamics is modeled as a second order, two degree of freedom vibration system. (assuming there is no vibration in the axial direction).

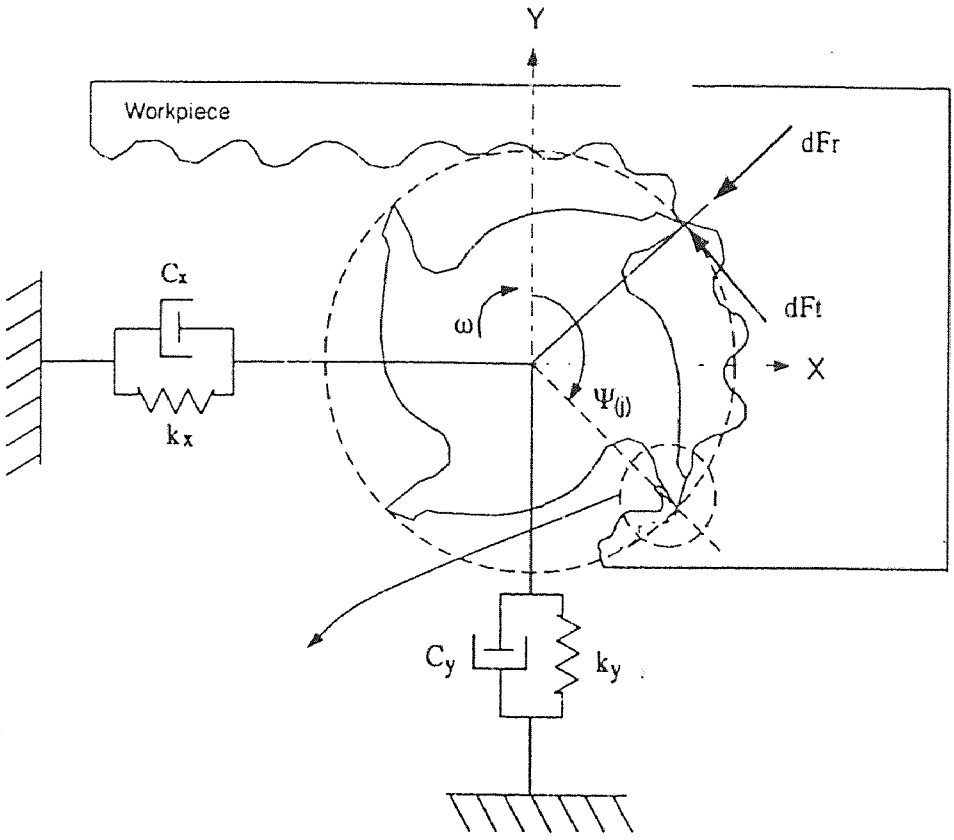



**Figure 3.6** System dynamics modeling

The modal parameters of the structure are experimentally determined and the differential equations of the vibration system are as follows:

$$\ddot{x}(t) + 2\xi_x \omega_{nx} \dot{x}(t) + \omega_{nx}^2 x(t) = \omega_{nx}^2 / k_x F_x(t)$$
$$\ddot{y}(t) + 2\xi_y \omega_{ny} \dot{y}(t) + \omega_{ny}^2 y(t) = \omega_{ny}^2 / k_y F_y(t) \quad (3.18)$$

By solving the above equation, the tool shifting position due to vibration at any time instance can be calculated.

Given the cutting force at time $t = t_0$ : $F_x(t_0), F_y(t_0)$ and the initial position of tool at this time instance: $x(t_0), y(t_0)$, $x(t), y(t)$ can be numerically solved from equation 3.18. The tool shifting position due to the vibration at the next time instance $t_0 + \Delta t$ is then also available. In the same way as we did for the deformation/deflection, the uncut chip thickness can then be updated based on the vibration of the cutter and thus the cutting force at time $t_0 + \Delta t$ can be updated.

The prediction procedure is quite similar to model III except in step e):

e) Accumulated cutting force at current instance $i$ is calculated. Substitute the cutting force into equation 3.18 and numerically solve the differential equations for $x(t), y(t)$. Tool shifting is then

$$\gamma_x(i+1) = x(t + \Delta t),\ \gamma_y(i+1) = y(t + \Delta t) \ .$$

Advantage: System dynamics is considered and modeled. Cutting force prediction is then based on the vibration of the tool and workpiece system. Also can be used for instability and chatter prediction and avoidance.

Disadvantage: Computational complexity is dramatically increased for numerical solution of the second order vibration equation.

## 3.4 Multipass Cutting Force Prediction with SDE Approach

### 3.4.1 Models Used in Our Research

The model accuracy and computation complexity are always in conflict with each other in cutting force prediction. As an application example for developing the object deformation used for deformed swept volume generation, using a relatively simple model as a demonstration illustration is quite reasonable. Furthermore, even we used the distributed cutting force models (II, III, IV), since cutting force is predicted vs. rotational angle (step length is 5 - 10°), the computation cost for generating grazing points at each of these rotation positions is formidable (There are thousands of rotations for just one cutting block). Therefore, in such a situation, we still have to use only average cutting force or maximum/minimum cutting force for the deformation integration to swept volumes.

The first model is therefore used for cutting force prediction in our current research. Although more complicated models (II, III, IV) can be integrated in the same manner, the computation cost will be dramatically increased. Both the tool deflection and deformation models discussed in section 3.2 are used for deformation generation.

### 3.4.2 Using Swept Volumes for Multipass Cutting Force Prediction

Used in equation 3.5 for the cutting force prediction, MRR is the key factor determining the cutting force. As in the ideal cutting situation, the chip cross section area $A$ is calculated as: $A = a * d$, where $a$ is radial depth of cut and $d$ represent axial depth of cut. But sometimes the machined surface error is so large that we can not neglect its

effect on the following machining process. For example, after a rough cut, the machined surface errors due to the tool deformation and/or the scallop are so large that we can not just use the simple ideal equation to calculate $A$. As shown in figure 3.7, the uncut chip geometry is different between the ideal surface and real machined surface after rough cut and thus the cutting force will also be different. For multipass cutting, especially for the rough cut followed by finish cut, we need to consider the effect of surface error after the previous cut (the rough cut) on current cut (finish cut).

As we can see, using swept volume representation, we have the deformed swept volume boundaries of the machining tools, which also represent the machined surface. Therefore, we can use the boundary of swept volume of previous cut pass as an input for MRR calculation and cutting force prediction of the current cut.

The basic process is as follows:

a) Generate the boundary of the deformed swept volume of previous cut $S_{\sigma_p}(P)$ which includes the tool deformation. (figure 3.7 (a))

b) Generate the undeformed swept volume of current cut $S_{\sigma 2}(P')$ which does not have the tool deformation. (figure 3.7(b))

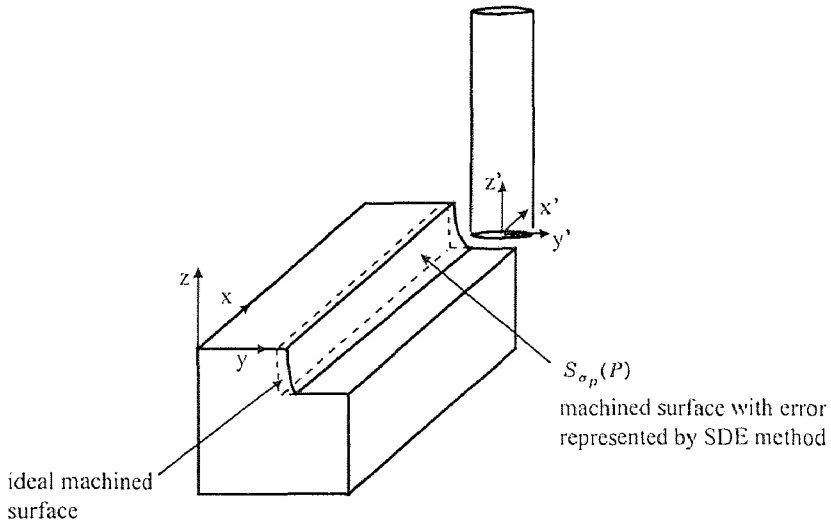
c) Calculate the chip cross section area from the above two swept volume boundaries:

   The area for uncut chip at time $t$ is shown in figure (c). Since the tool is discretized slice by slice for programming, the cross section area can be calculated by adding discrete areas together:

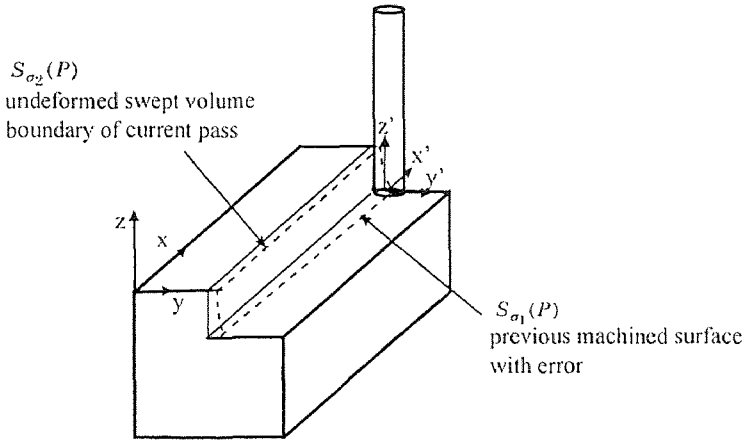
$$A = \sum_{z=1} r(x,y,z,t) \cdot dz \tag{3.19}$$

where $r(x,y,z,t) = |\vec{P}(x,y,z,t) - \vec{P}'(x,y,z,t)|$ and $\vec{P}$, $\vec{P}'$ represent the vector pointed to grazing points $(p, p')$ on the boundary of deformed swept volume of previous cut and undeformed swept volume of current cut, respectively. Using equation 3.5 & 3.1,3.3, predict the cutting force and tool deflection/deformation.

d) Using the predicted tool deformation/deflection in step c), regenerate the deformed swept volume of current cut.

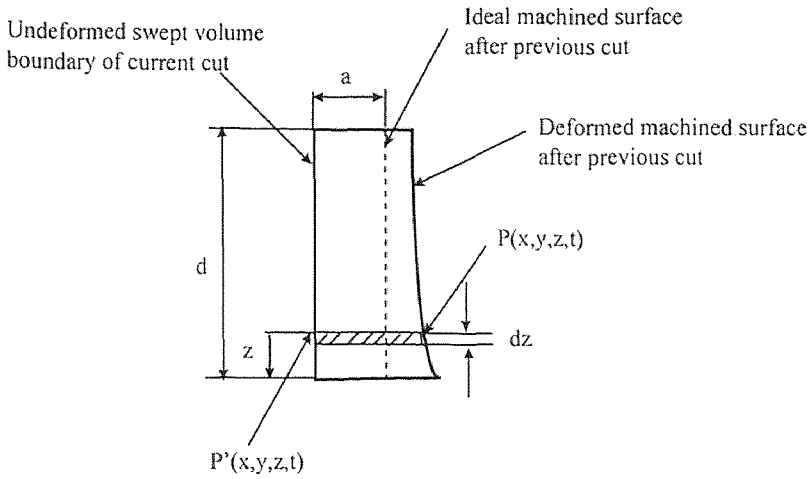



(a) Machined surface after previous cut pass




(b) Undeformed swept volume of current pass

**Figure 3.7** Cutting force prediction for multipass cut




(c) Uncut chip cross section geometry

**Figure 3.7** (continued) Cutting force prediction for multipass cut

In this way, we can use the swept volume approach not only for geometric cutting simulation, but also for more accurate calculation of the uncut chip cross section which is used for multipass cutting force prediction. This algorithm is also implemented in our program for multipass NC simulation.

# CHAPTER 4

# IMPLEMENTATION AND APPLICATION IN NC SIMULATION AND VERIFICATION

As we discussed in chapter 2, the SDE method can be extended to include object deformation to represent the deformed swept volumes. The swept volumes encountered in manufacturing automation are always subject to some deformation. Given the module to predict and calculate the tool deformation and the module to generate the deformed swept volume, we can generate the deformed swept volume of the end mill in machining process and predict/simulate the machined surface error.

A deformation calculation program was developed in this project to predict the end mill deformation. An SDE module which integrates the deformation was also developed. The deformed swept volumes of the end mill in machining then were generated. By integrating with ProEngineer, the mill swept volume then was visualized and subtracted from the workpiece to simulate the machined part. Machined surface errors were also analyzed and the surface patches where the error exceeded tolerance were indicated.

## 4.1 Tool Motion Generation

Before applying the SDE with deformation to NC simulation and verification, several preliminaries and approaches need to be discussed first. One is the machine and tool motion model and path generation. CL data, which indicates the cutter location, contains the information of the tool tip position and tool orientation in machine coordinate frame.

Since most of the CAD/CAM systems supply the CL data generation, we use the CL data to generate tool motion equation by assuming the linear interpolation of machine.

We generally assume that the multi-axis machine has the joint-interpolation motion. The linear and circular interpolation are the most commonly used methods. Some other methods such as parabolic and cubic interpolation are available on some machines. Although each kind of interpolation can be implemented in the SDE motion equation, we assume linear interpolation in our project since it is the most common machine motion interpolation method.

The CL data basically contains the information of the tool position and orientation in the machine coordinate frame. One CL datum contains the tool position and orientation information which is expressed as $(x_c, y_c, z_c, i_c, j_c, k_c)$, where $(x_c, y_c, z_c)$ represent the tool tip position in machine coordinate frame and $(i_c, j_c, k_c)$ stands for the normal cosine values of the spatial angle of the tool axis vector in machine coordinate frame.

For a given tool position defined by a CL datum $(x_c(t), y_c(t), z_c(t), i_c(t), j_c(t), k_c(t))$ as shown in figure 4.1, the transformation matrix of the tool frames from the machine coordinate frame can be express as:

$$\begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix} = \begin{pmatrix} x_c(t) \\ y_c(t) \\ z_c(t) \end{pmatrix} + R \cdot \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \tag{4.1}$$

where $(x_0, y_0, z_0)$ is the initial tool tip position.

R is rotational transform matrix

Although different machines have different motion types, generally we assume the roll and pitch motions of the tool according to general coordinate system. As we can see,

the tool axis vector z, whose orientation is defined as $(i_c(t), j_c(t), k_c(t))$, can be transformed from z axis by rotating α angle about x axis and then rotating β angle about y axis. Therefore, the R matrix can be defined as:

$$R = \begin{pmatrix} \cos\beta & -\sin\alpha\cdot\sin\beta & \cos\alpha\cdot\sin\beta \\ 0 & \cos\alpha & -\sin\alpha \\ -\sin\beta & \sin\alpha\cdot\cos\beta & \cos\alpha\cdot\cos\beta \end{pmatrix} \tag{4.2}$$

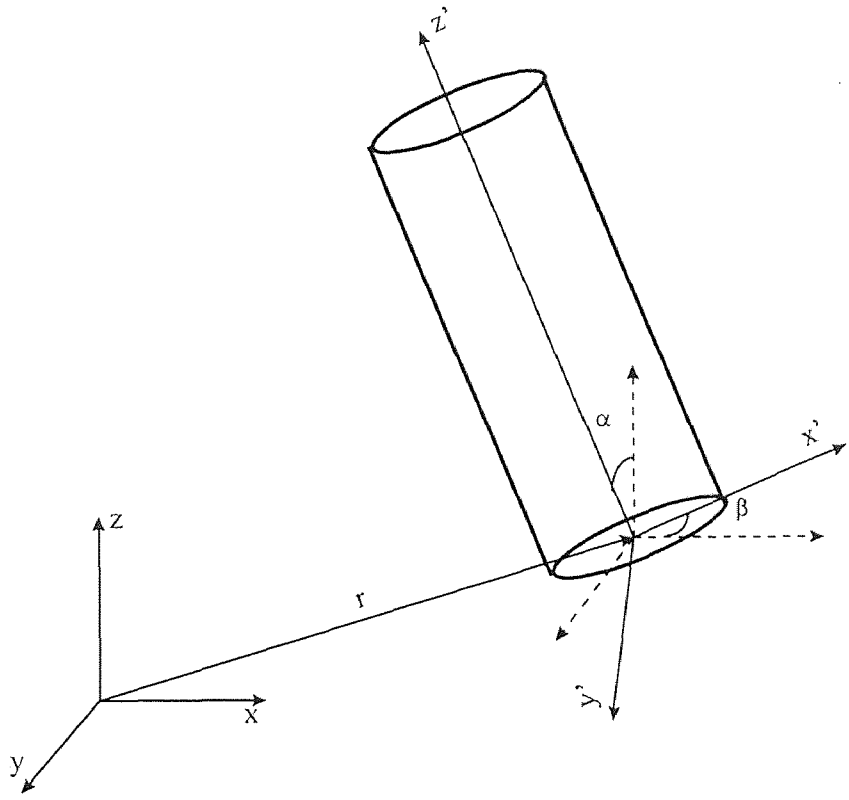



**Figure 4.1** Coordinate Frames Transformation

By solving the following equation for α, β:

$$\begin{pmatrix} \cos\beta & -\sin\alpha\cdot\sin\beta & \cos\alpha\cdot\sin\beta \\ 0 & \cos\alpha & -\sin\alpha \\ -\sin\beta & \sin\alpha\cdot\cos\beta & \cos\alpha\cdot\cos\beta \end{pmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} i(t) \\ j(t) \\ k(t) \end{bmatrix} \tag{4.2}$$

We obtain

$$\begin{aligned} \alpha &= -\tan^{-1}(j_c(t)/\sqrt{i_c^2(t)+k_c^2(t)}) \\ \beta &= \tan^{-1}(i_c(t)/k_c(t)) \end{aligned} \tag{4.3}$$

Given CL data for one block cut

$(x_c(0),\ y_c(0),\ z_c(0),\ i_c(0),\ j_c(0),\ k_c(0)) \rightarrow (x_c(1),\ y_c(1),\ z_c(1),\ i_c(1),\ j_c(1),\ k_c(1))$

assuming linear interpolation, the interval CL data can be calculated from the initial tool position and the final position:

$$\begin{pmatrix} x_c(t) \\ y_c(t) \\ z_c(t) \\ i_c(t) \\ j_c(t) \\ k_c(t) \end{pmatrix} = \begin{pmatrix} x_c(0) \\ y_c(0) \\ z_c(0) \\ i_c(0) \\ j_c(0) \\ k_c(0) \end{pmatrix} + \begin{pmatrix} x_c(1)-x_c(0) \\ y_c(1)-y_c(0) \\ z_c(1)-z_c(0) \\ i_c(1)-i_c(0) \\ j_c(1)-j_c(0) \\ k_c(1)-k_c(0) \end{pmatrix} \cdot t \qquad where \quad t \in [0,\ 1] \tag{4.4}$$

Substituting $(x_c(t),\ y_c(t),\ z_c(t),\ i_c(t),\ j_c(t),\ k_c(t))$ into the translational and rotational transform matrix in above, we can calculate the transformation equation at any time $t \in [0, 1]$ for this cutting block

## 4.2 Programming and Integration with CAD/CAM System

As the SDE algorithms and tool deformation calculation presented before, a program for the generation of the deformed swept volumes was developed. Also, the integration with a commercial CAD/CAM software package (Pro/Engineer) was introduced. The basic process of program integration is as follows:

1) We use Pro/Engineer for the machined part design; Pro/Manufacturing for CL data generation;
2) Use the generated CL data and manufacturing parameters as inputs to our deformed swept volume program; Calculate cutting force and deformation at each section; Generate deformed swept volume block by block (one block means from one CL data to another);
3) The output of our program (deformed swept volume boundary points, organized as Pro/E readable file) is then input back to Pro/Engineer for visualization and Boolean subtraction from the workpiece for material removal and surface error checking.

The details of programming and integration of the physical deformation with the SDE method to generate deformed swept volumes of end mill can be described as the following steps:

I. **CL data generation**:

Create designed part in Pro/Engineer;

Setup workpiece, select tool and manufacturing parameters;

Use Pro/Manufacturing to generate CL data;

Output CL data sequence by sequence as *.ncl.

II. **Read in CL data file:**

Read in *.ncl file by file;

Abstract the CL data (the data of cutter location, which is after "GOTO" in *.ncl) block by block.

III. **Generating tool motion equation**:

Using CL data block by block :

$$(x_c(n-1), y_c(n-1), z_c(n-1), i_c(n-1), j_c(n-1), k_c(n-1))$$
$$\rightarrow (x_c(n), y_c(n), z_c(n), i_c(n), j_c(n), k_c(n))$$

Transform CL data to generate tool motion equation for each block: (As discussed in section 4.1)

**IV. Input simulation parameters:**

Tool selection: projected length $L$; diameter $D$; number of flute $m$; material $E$

Manufacturing setup: spindle speed $n$; feedrate $f$; depth of cut $d$

Machine parameter: specific power $P_{sp}$; deflection parameters $E_r, E_h$

**V. Discretizing:**

discretize tool into K slices; Determine time sections number TT during one cut block;

**VI. Cutting force prediction**:

Use the swept volume boundary of previous cut as input (Multipass simulation) ?

NO: 1) Calculate axial depth of cut $d$ according to manufacturing setup and current CL data

2) Calculated radial depth of cut $a$

3) Calculate cutting force according to equation 3.5

YES: 1) Generate undeformed swept volume of current cut block (by setting $D(x,t) = 0$)

2) Choose the swept volume boundary of previous cut as input for current MRR calculation.

3) Calculate the chip cross section area according to equation 3.19

4) Calculate the cutting force at each time instance (section) during current cut

**VII. Calculate tool deflection/deformation at each time instance** according to equation 3.1, 3.3

**VIII. Generate the boundary of deformed swept volume:**

1) Using SDE with deformation (Equ.2.15) and extended BFF (Equ. 2.17) to calculate the tangency function and identify the grazing points for each slice of the tool at each time instance $t$ .

2) Generate the data of the boundary of swept volume, both the deformed swept volume and undeformed swept volume ( $D(x_0,t) = 0$ );

3) Compare the boundary of deformed swept volume of end mill with undeformed swept volume for boundary error;

4) Record the boundary points where the deformation exceeds the tolerance of machine error.

**IX. End of current cutting pass?**

Yes: continue;

No: goto step II.

**X. Material removal simulation**:

1) Organize grazing points section by section as closed section curve;

2) Output grazing points as Pro/Engineer readable file format: *.ibl to construct swept volumes; (We can input formatted point data to Pro/Engineer to construct curves and/or solid. Details please refer to Pro/Engineer manual)

3) Use "cutout" function in "Pro/Assembly" to cut swept volume from workpiece;

4) For visualizing the deformation, show the surface patch composed of the boundary points where the deformation exceed the tolerance.

**XI. End of reading all CL data file?**

No: go to step II;

Otherwise: end of programming.

**XII. Need to modify manufacturing setup to reduce error to within tolerance?**

Yes: Goto step I;

No: End of simulation.

As the programming and integration procedure discussed above, a flow chart of the programming and integration is shown in following:

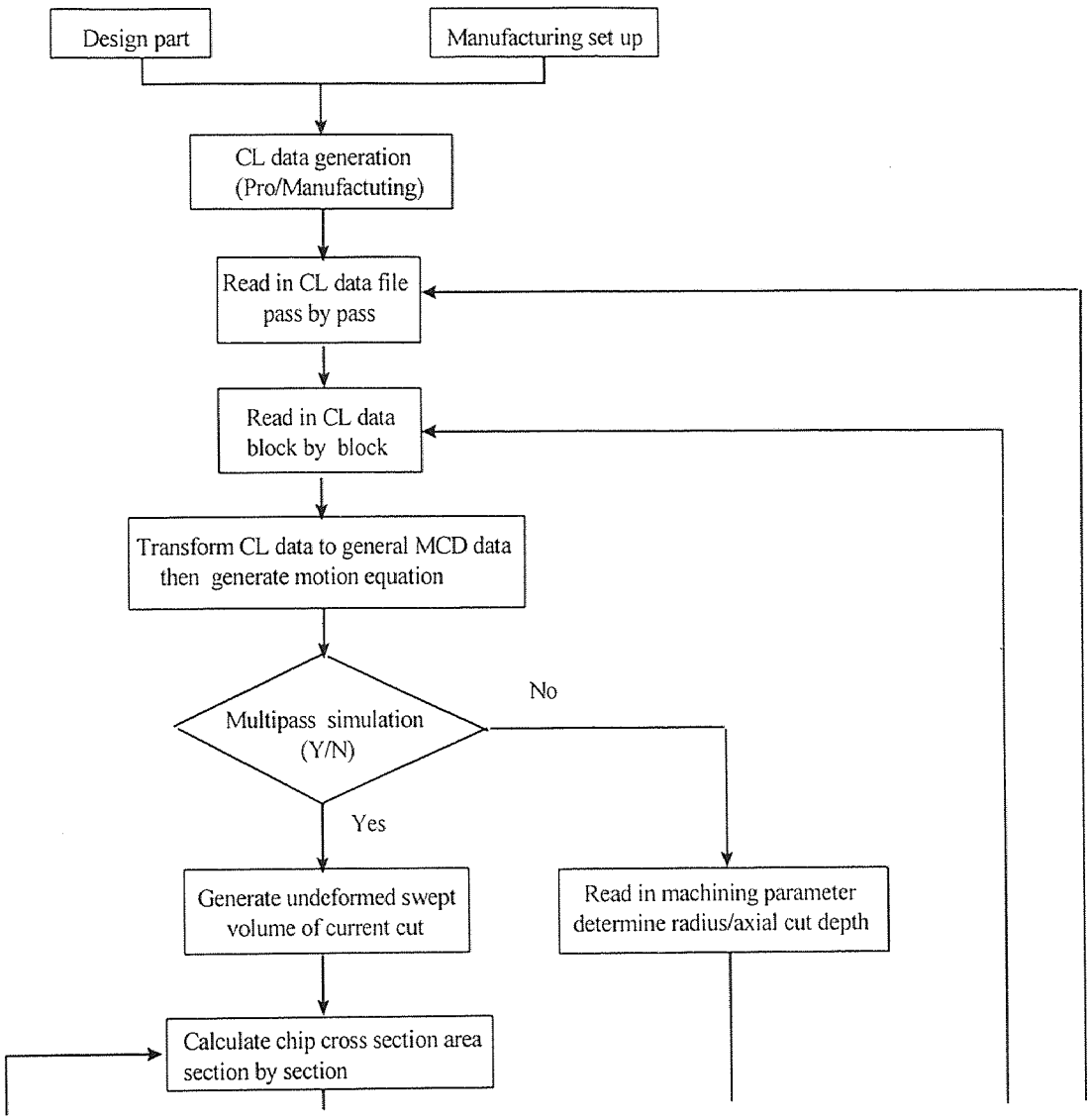



**Figure 4.2** Programming and Integration Scheme

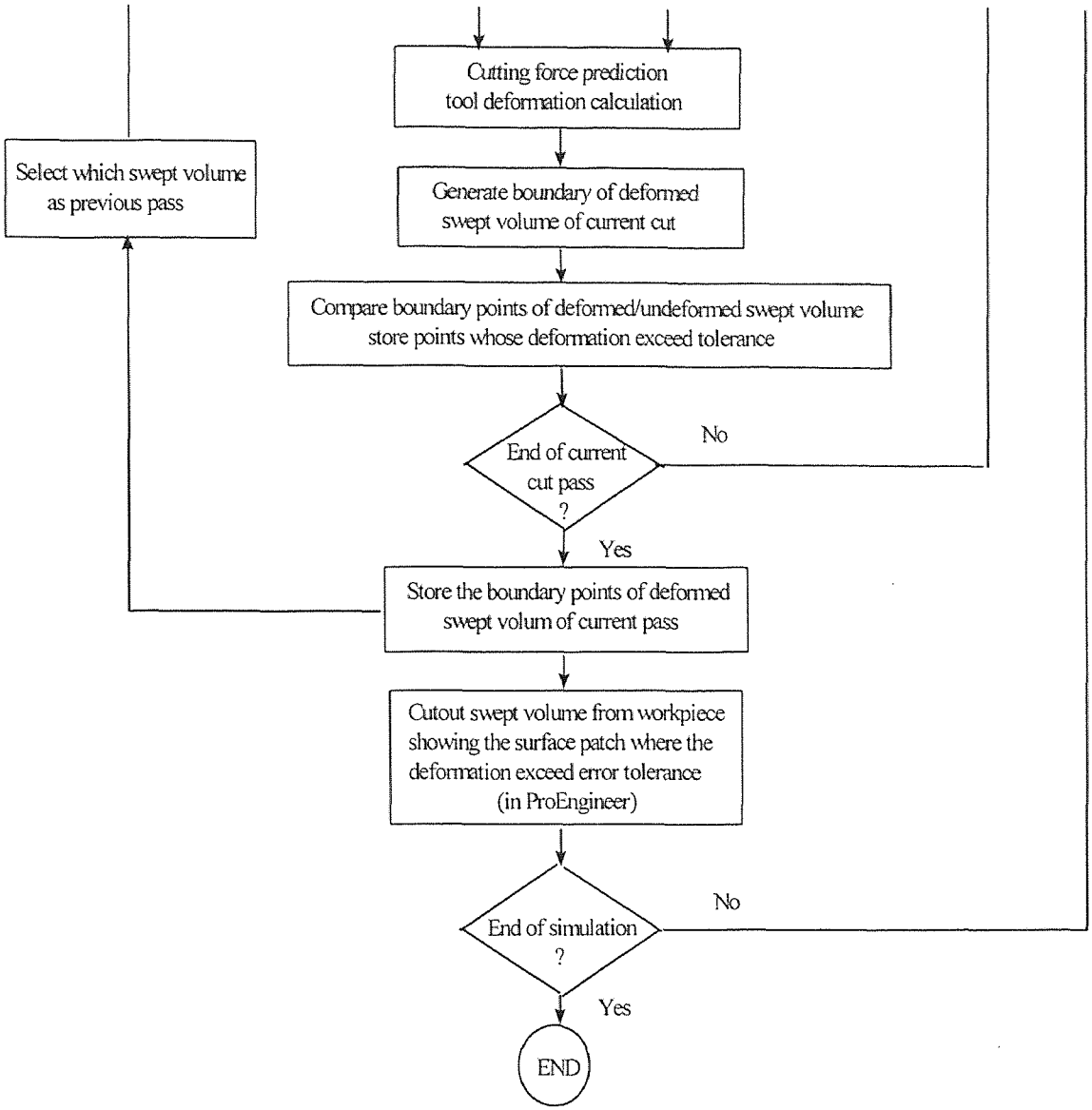



**Figure 4.2** (continued) Programming and Integration Scheme

## 4.3 Another Approach for Cutting Force Prediction

As we discussed before, there are several more accurate models of cutting force prediction used by many researchers. Actually, much research has been done and some cutting force prediction programs are available. For example, a software package call

"Ballend", which enables the user to input manufacturing parameters and predict cutting force and surface quality, is available from Altintas's group at the Univ. of British Columbia. A comprehensive program is also available from DeVor, University of Illinois at Urbana Champaign. This software, "EMSIM", is capable of simulating several typical cutting geometries such as step cut, slot cut, ramping cut as well as corner cut. Although they can not simulate continuously for more complex geometry, the users can break a geometry into the combination of these typical geometries.

Although the integration of the cutting force and deformation from these program with SDE is basically the same as the build_in cutting force prediction which we discussed before, some problems arise:

1. They can not simulate part of more complex geometry. Therefore, we may need to break a geometry into the combination of these typical geometries.
2. There is too much cutting force information (usually, the step of simulation is $5^{\circ}$ of rotation) for our swept volume generation, and computation time is formidable.

The recommended simulation step is less than 10 degrees to obtain acceptable cutting force simulation results. On the other hand, however, if we choose the same step length to calculate the grazing points as the step for force simulation, we will have to calculate millions of grazing points. For example, we simulate a 2-flute mill with 0.01mm/tooth feedrate. We want to generate the swept volume of the mill moving within a block: 10mm, which is 500 rotations ( since 10/(0.01x2)=500 ). Assuming the step length of the cutting force simulation is 10 degrees and we use the same step length for

calculating the grazing points, we need to calculate 18,000 sections (500x360/10=18,000) of grazing points which have 18,000x80 points (assuming tool is divided into 40 slices). It is very huge and even formidable for just one block cut. Therefore we need to optimize the input cutting force to extract most useful information for our SDE implementation purpose.

The approach we adopt here for optimizing read in cutting forces is as following.

For rough cut, we concern more about the materials left on the machined surface which will be removed in the following cut (such as finish cut), rather than the machined surface quality after rough cut. Therefore, it is quite reasonable that we read in the cutting force and calculate the average cutting force during each tool rotation for our deformed swept volume generation. As shwon in Fig. 4.3(a), the material will be removed can be approximated by calculating the area between the average deformation and the tool contour. The real material will be removed in finish cut is very close to the approximated area. In this way, we can dramatically reduced the deformed swept volume calculation time by 36 times.

However, for the finish cut, we concern about the machined surface quality. That means the averaging surface error is not enough for surface quality checking. Therefore, we read in the maximum and minimum cutting forces from the simulated forces and calculate deformed swept volume accordingly. As shown is Fig. 4.3(b), the surface error wave lies inside the maximum/minimum deformation lines. We make sure the most important information of surface errors such as maximum/minimum deformations are indicated in our simulation.

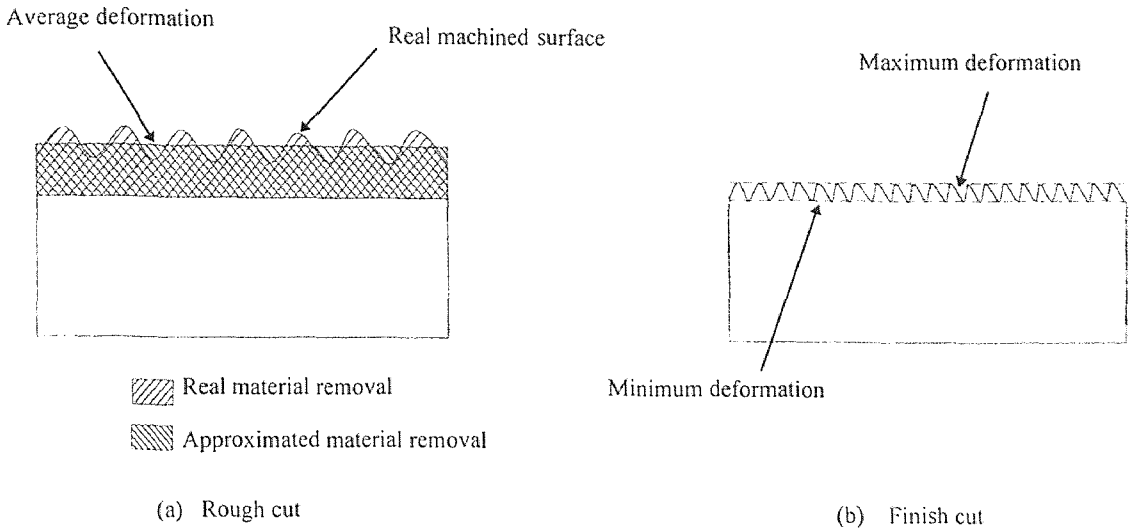



**Figure 4.3** Cutting Force/Deformation Approximation

However, another problem arises by this approximation. As we can see from the SDE equation with deformation:

$$X_\sigma = \dot{x}(t) = \dot{\xi}(t) + \dot{B}(t)x_0 + \partial_t D(x_0, t)$$

if we only use the maximum/minimum cutting forces to calculate the deformation, we do not have enough data to calculate $\partial_t D(x_0, t)$ by numerical method. If we only have the maximum/minimum cutting forces, we calculate the $\partial_t D(x_0, t)$ by the first order forward difference method:

$$\partial_t D(x_0, t) = W(x_0) * \partial_t F(x_0, t) = W(x_0) * [F_{\max} - F_{\min}] / (t_{\max} - t_{\min})$$

which is too rough and there will be some error between the approximated $\partial_t D(x_0, t)$ and real $\partial_t D(x_0, t)$.

One way to improve the approximation is descried as following:

1) Abstract the maximum/minimum cutting forces

2) Read in the cutting forces which are close to the maximum/minimum forces

3) Calculate $\partial_t D(x_0,t)$ through all these cutting force data by numerical method.

There are several different numerical methods for calculating the differentiation, for example Difference method, Lagrange's Interpolation method, Newton' Interpolation formula, etc. Here we use the three-point forward numerical differentiation formula which is relatively simple for calculating derivatives from data points:

$$y'_x(x_0) \approx \frac{1}{h}[\frac{1}{2}y(x_0-h) - 2y(x_0) + 1\frac{1}{2}y(x_0+h)] \tag{4.5}$$

By substituting y with deformation, we can derive the equation:

$$\partial_t D(x_0,t_m) \approx \frac{1}{\Delta t}[\frac{1}{2}D(x_0,\ t_m-\Delta t) - 2D(x_0,\ t_m) + 1\frac{1}{2}D(x_0,\ t_m+\Delta t)] \tag{4.6}$$

where $\Delta t$ is the time interval between $t_m$ and $t_{m+1}$

# CHAPTER 5

# SIMULATION EXAMPLES

The following examples illustrate how to use our SDE program to generate deformed swept volumes and apply them in NC simulation and verification.

In example 1, a ramping cut process is simulated by two ways: one, using cutting force simulation software "EMSIM" (which is developed by DeVor's group) for the cutting force prediction; and the second, using our build_in cutting force prediction to simulate deformation and cutting process. The two approaches are then compared and discussed.

Another example is a complex milling for a mold which is part of a mouse shell mold. The whole machining process is simulated to include rough cut and finish cut. To reduce machining error to within tolerance, a modification of finish cut is suggested and also simulated.

Since the deformation is so small that we can not see the deformation, two ways are used for the deformation visualization: one is to amplify the deformation; another is to show the surface patch (in red color) where the deformation exceeds surface error tolerance.

## 5.1 Example 1

A ramp cut process is simulated in this example. Two ways of cutting force prediction are used: one is to use cutting force simulation software for the cutting force prediction; the other is to use our build_in cutting force prediction to simulate the deformation and

cutting process. The two approaches are compared and discussed. We used the simulation program "EDSIM" developed by Devor' s research group for outside cutting force prediction.

The tool initial and final positions with the designed part are shown in figure 5.1

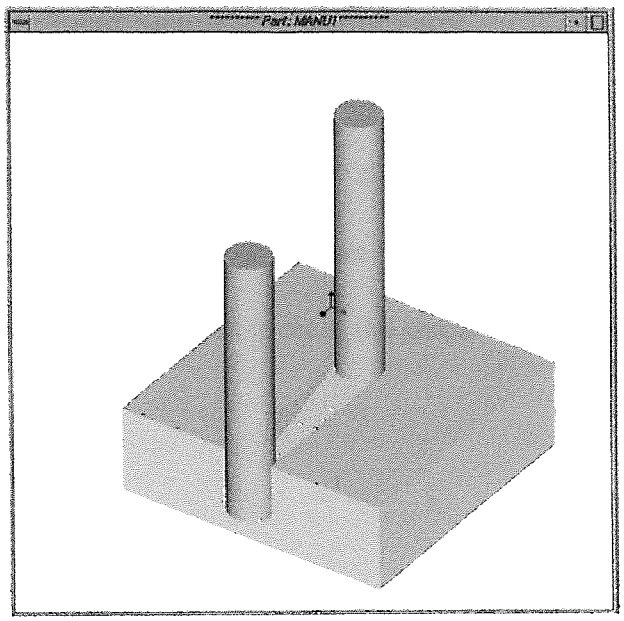


**Figure 5.1** Tool initial and final positions in example 1

**Simulation parameters**:

| | | |
|---|---|---|
| Work piece data: | Material: 1018 steel | |
| Tool parameter: | Tool type: | Flat_end mill |
| | Diameter: | 12.7 mm |
| | Projected Length: | 76.2 mm |
| | Number of flutes: | 4 |
| | Helix Angle: | 30 degree |
| Manufacturing Parameter: | Cutting style: | Ramp cut |
| | Entry axial Depth of cut: | 10 mm |
| | Ramp angle: | 20 degree |
| | Cut_step: | 12.7mm |

Feed per tooth: .02mm
Spindle speed: 500 rpm

Simulation parameter: Cutting Force : Optimized
CL data (3 axis mill): (0.000000, 0.000000, -10.000000)
(27.47475, 0.000000, 0.000000)
Surface error tolerance: 250 um

### 5.1.1 Approach One: Input Simulated Cutting Force

We read-in the simulated cutting force which is generated from the EMSIM software. Cutting force input was optimized as we discussed in section 4.3 to reduce the computation time. Only maximum and minimum forces during each rotation are read-in. The simulated cutting forces are read in from the output file of EMSIM and sketched in figure 5.2

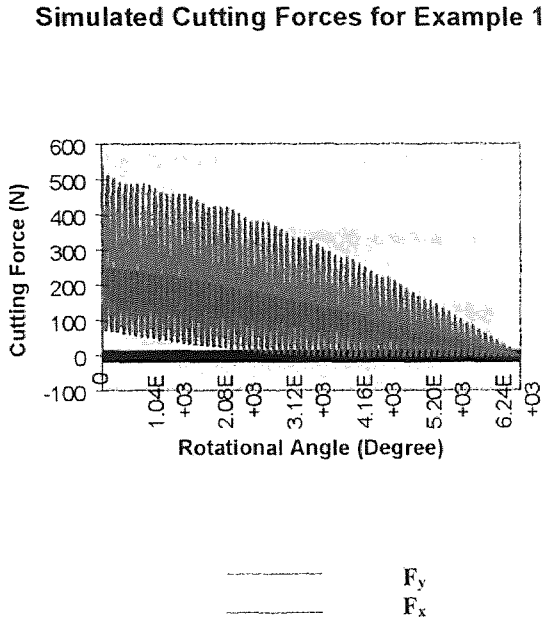



**Figure 5.2** Simulated Cutting Forces

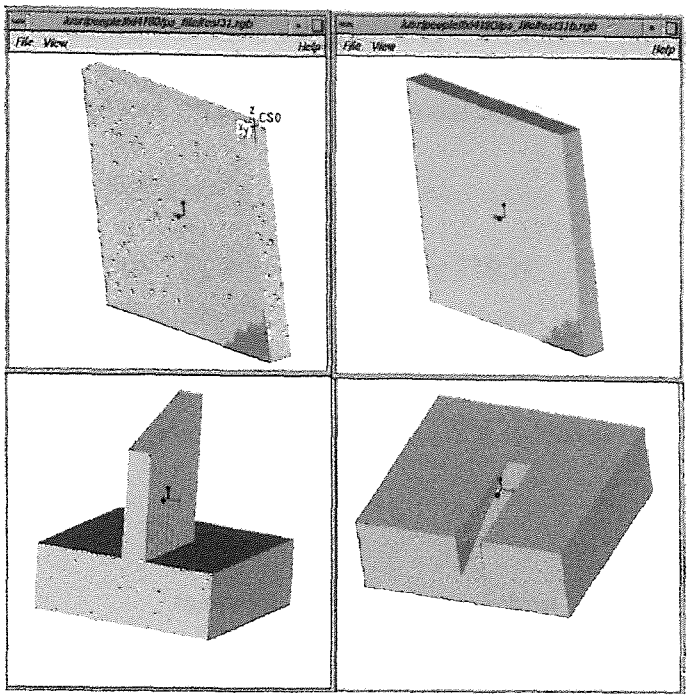
Cutting forces are then read into the SDE program (Refer to APPENDIX B final_cl.c) to generate the deformation and deformed swept volumes. The grazing points are output in Pro/Engineer readable file format *.ibl to Pro/E for visualization and Boolean subtraction from the workpiece. Read_in cutting forces and deformations are also output as text file for detail checking.

The major results of NC simulation and verification:

Surface error tolerance: .25 mm
Maximum cutting force: $F_y = 519.7$ N; $F_x = 237.4$ N
Maximum tool deformation: .301 mm
Computation time: approximately 1' 20"

Picture a in figure 5.3 shows the boundary of the swept volume with the red surface patch which indicates the area where the deformation exceeds setup tolerance .25 mm. Picture b. represents the swept volume of the mill. Pictures c and d are the material removal process and simulated machined part, respectively.

As we can see from the simulated machined surface, some area of the surface has larger deformation than the tolerance, which we set as 250um. The reason is because the ramp cutting has larger cutting forces in the beginning portion of the machining which results in more deformation. Therefore, modification of the manufacturing setup is suggested such as changing the axial depth of cut and/or the radial depth of cut.




| a | b |
|---|---|
| c | d |

**Figure 5.3** Ramp Cut Simulation with Approach One

### 5.1.2 Approach Two: Using Build-in Cutting Force Simulation

In the second approach, the cutting force simulation is already integrated into our program (as discussed in section 4.3). The manufacturing setup and parameters are the same as approach one.

The depth of cut is changing in the ramping cut (also for some other applications). Here we use the coordinate system to determine the axial depth of cut in program:

We set the global (workpiece) coordinate system on the top corner of the workpiece in Pro/Manufacturing. That means the CL data is generated based on this coordinate

system. We also choose it as our SDE programming coordinate system. Therefore, the z coordinate of the CL data (tool tip position) is the axial depth of cut.

Simulation and verification result:

Maximum Cutting Force: $F_y = 437.99$ N; $F_x = 219$ N
Maximum deformation: .2637 mm
Computation time cost: approximately 20"

Predicted averaging cutting forces were output and sketched in Figure 5.4:

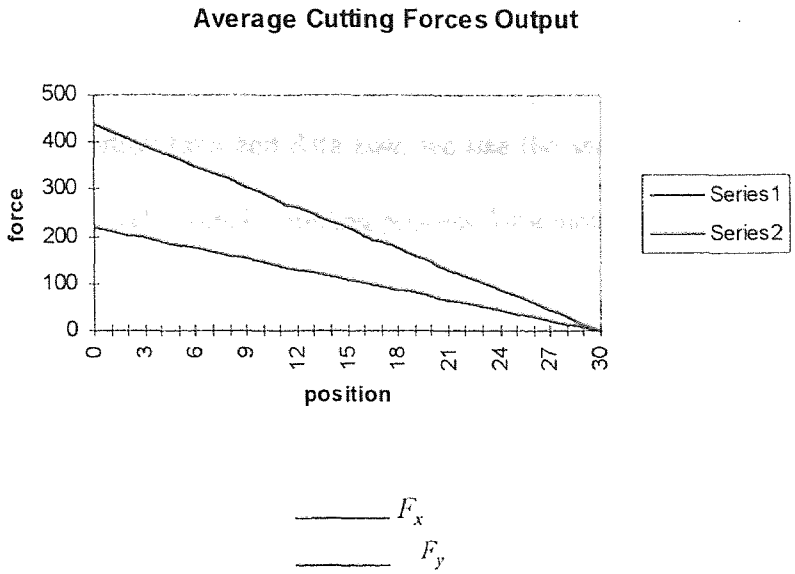



**Figure 5.4** Predicted Average Cutting Force

### 5.1.3 Compare the Two Approaches:

As we can see from the above simulation results, the major difference between these two approaches for the deformed NC simulation is accuracy and computation time. There is some trade-off between the simulation accuracy and computation time.

The averaging cutting force has no details of the waviness of the actual cutting force and is approximately 10%~20% less than the simulated maximum cutting force. Therefore, if we want to check out the details of the cutting force and deformation, the second approach does not supply enough information. However, the computation time of the first approach is 5 times as much as the second approach and even takes much more time for visualization in Pro/E. Since the grazing points is much more than in the second approach, it take almost 7 times as long for visualization. The computation time cost and the huge size of the grazing points make the first approach unsuitable for multi-block cutting simulation and even formidable for a real machining simulation. Therefore, considering the computation time and data size, we use the second approach for another example, which is a relatively complex milling process for a mouse shell mold.

## 5.2 Example 2

To illustrate the SDE approach for the deformed swept volume representation and its potential on dynamic NC simulation and verification, a more realistic and complicated example is given. This is a mouse shell mold milling simulation. The mold for mouse was designed in Pro/Mold design. We use Pro/Manufacturing to generate the CL data. Two NC sequences are used. First is rough milling for material removal; second is trajectory milling along the mold side for finish milling. Manufacturing setup and simulation selection are as following:

| | NC sequence 1 | NC sequence 2 |
|---|---|---|
| Workcell: | 3 axis mill | 3 axis mill |
| Milling type: | trajectory | trajectory |
| Feed rate: | 5 ipm | 5 ipm |
| Spindle speed: | 500 rpm | 700 rpm |
| Axis depth: | changing (one step) | changing (one step) |

| | | | |
|---|---|---|---|
| Stepover: | 0.2" | | None |
| Tool: | ¼ " (HS) | | 1/8 " (HS) |
| Workpiece material: | 7075 aluminum | | 7075 aluminum |
| Multipass simulation: | No | | Yes |
| Surface error tolerance: | | 0.0022" | |

A snapshot of the CL data generating in Pro/Manufacturing is shown in figure 5.5. We have a small piece of program for the CL data abstraction from the Pro/E generated CL data file and transferring to MCD as we discussed in section 4.1. The Pro/E generated CL data file is *.ncl. By reading in this file into program "cl.c" (Refer to APPENDIX A), the CL data are then abstracted and output to SDE program for motion generation.

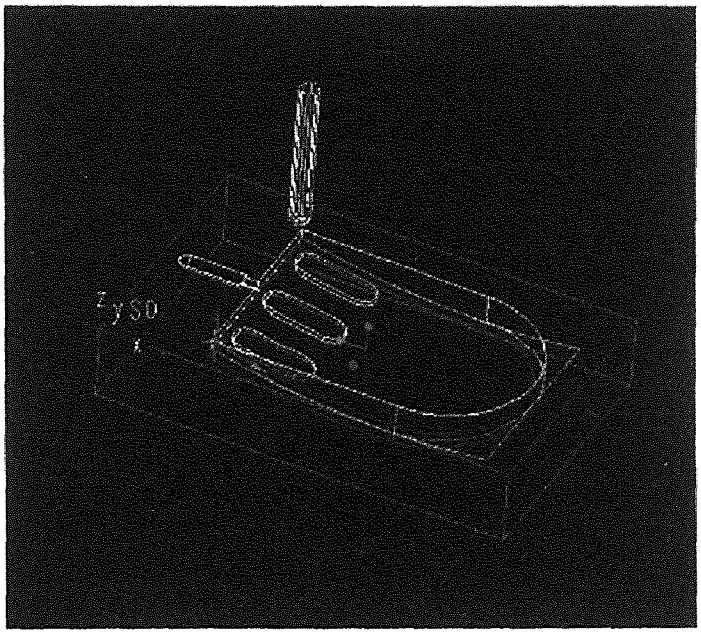


**Figure 5.5** CL data generation in Pro/Manufacturing

For rough cut, since the radius cut depth (0.2") is much larger than the machining error (tool deformation), we do not use multipass simulation. For the finish cut, machined surface error after rough cut is used as the input (multipass simulation). Simulation is shown step by step in the following:

For illustrating of the deformation, we amplified the deformation by 2.5 times. We can see the difference between deformed swept volume and undeformed swept volume in figure 5.6. The boundary in blue color is the boundary of undeformed swept volume of this cut pass. The points (curves) in yellow color indicate the boundary of deformed swept volume.

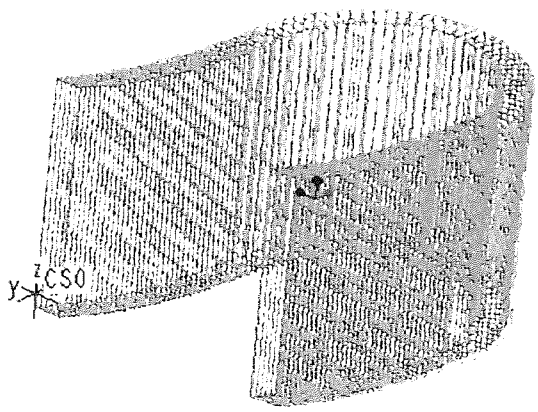



**Figure 5.6** Boundary comparison between deformed and undeformed swept volume

The NC simulation process is described as following:

1. Read in Pro/E generated CL data file into "cl.c" (See appendix A) for CL data abstraction and transferring to MCD
2. Input the CL data to the SDE program block by block for deformed swept volume generation
3. Output the grazing points as Pro/E readable *.ibl file. A sample of this file is show bellow.

   / This is a sample file format for *.ibl /

```
closed
arclength
begin section ! 1
begin curve ! 1
  -0.009211      6.33739        -10.0000
  -0.015697      6.33643        -11.9057
                    .
                    .
begin curve ! 2
                    .
                    .
```

4. Input the file *.ibl into Pro/E (in "Part" – "Adv. Feature" – "Read in from file")
5. Assemble the workpiece and swept volume together (NOTE: the coordinate system must be the same as in Pro/Manufacturing)
6. Use the "Cutout" function in Pro/Assemble to cut the swept volume from workpiece
7. Generate the surface patch from another output file from our program "surface.ibl" which records the grazing points whose deformation exceed the tolerance. Output into Pro/E ("surface"—"Adv. Feature" – "Read in from file"). Put red color to this surface patch to indicate the area where the deformation exceeds the tolerance.

Figure 5.7 shows the machining process of rough cut.

(a) is the snapshot of first pass assembly in Pro/Assembly;
(b) is the workpiece after material removal (cutout the deformed swept volume). Red surfaces indicate that surface error exceeds tolerance;
(c) is the second pass with stepover 0.2";
(d) is the workpiece after the rough cut.

As we can see, almost all of the profile surfaces have the errors larger than tolerance. Therefore, we need a finish cut .

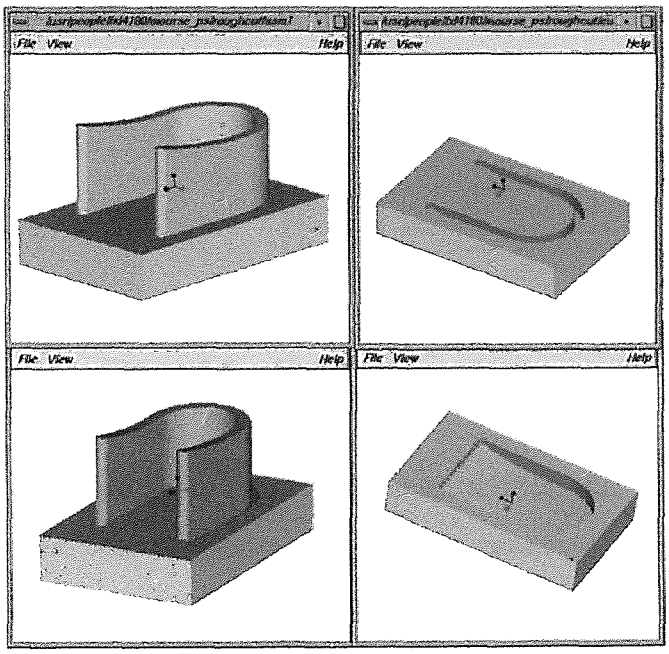
The same procedue for the finish cut. Figure 5.8 shows the process of finish cut.

(a) is the undeformed swept volume boundary points;

(b) is the swept volume without deformation;

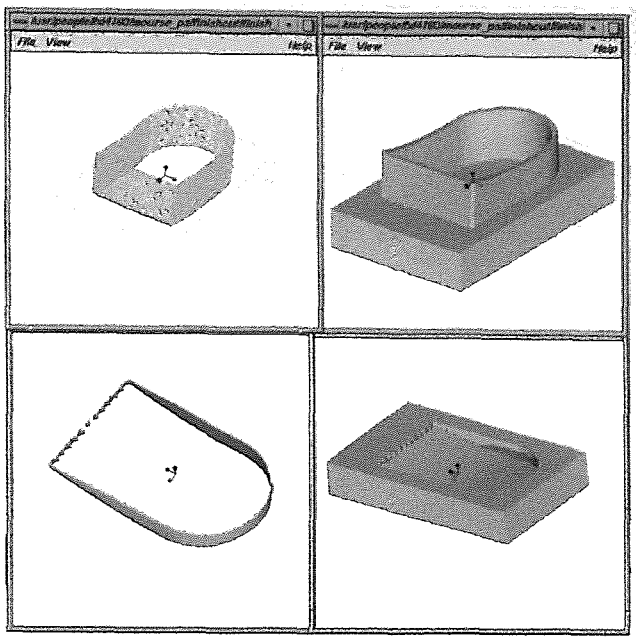
(c) shows the material which will be removed in finish cut ( composed by the machined surface after rough cut and the boundary surface of undeformed swept volume of current cut shown in (a)).

(d) shows the machined part with deformation.




| a | b |
|---|---|
| c | d |

**Figure 5.7** Rough cut simulation



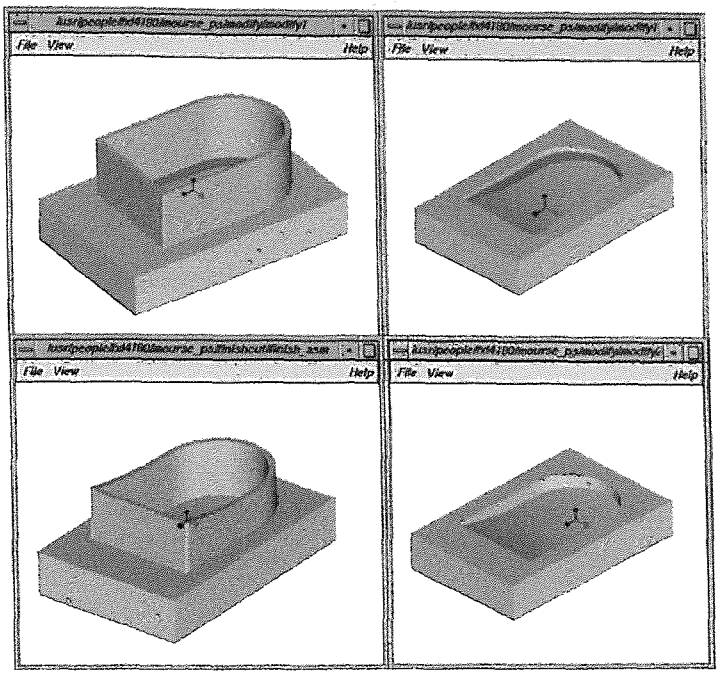
| a | b |
|---|---|
| c | d |

**Figure 5.8** Finish cut simulation

As we can see, some areas of the machined surface still have error exceeding tolerance. Therefore, we need to change the machining parameter or even change the NC sequence. One possible change: reduce the cut depth of finish cut to reduce MRR axial depth of cut: .20" .

By this modification, the machining process of finish cut is separated into two steps. Use Pro/Manufacturing to generate CL data again and input to our program.

Machining process is simulated again according to the manufacturing modification (figure 5.9). There is no surface error indicated as exceeding tolerance. Therefore, this modification is acceptable to reduce the machining error.




| | |
|---|---|
| a | b |
| c | d |

**Figure 5.9** Modified finish cut

# CHAPTER 6

# CONCLUSION AND REMAKS

## 6.1 Conclusions

We have demonstrated the generation of deformed swept volumes with the SDE approach and its potential applications in NC simulation and verification. Major results include:

- SDE algorithms were extended to include general spatial deformation
- End milling process was analyzed, including the calculation of cutting force and tool deformation
- A program was developed for deformed swept volume generation
- Implementation and potential applications in NC simulation & verification were discussed with examples given for cutting simulations.

Several difficulties and solutions were discussed about the implementation of SDE with deformation and applications in NC verification. Based on this approach, a program was developed for the generation of deformed swept volumes of machining tools. The linear and nonlinear deformation due to the cutting forces of the tools were calculated and integrated in SDE program.

As we have demonstrated in this project, the SDE method can be extended to include general spatial deformation, which is the tool deformation in machining, arm deflection in robotics, etc. This approach can be used in NC simulation and verification which not only can enable visual checking, but also can allow numerical verification such as surface error checking.

Although we used a simple model for our build_in cutting force prediction, we also demonstrated the capabilities of integration with the results of the more accurate and complex models for cutting force prediction. By generating the deformed swept volumes and using their boundary for the more accurate chip cross section calculation of the following cut, we obtain a more accurate way for the prediction of cutting force and deformation, especially, machining process of rough cut followed by finish cut. The deformed swept volumes with different tool and manufacturing parameters were generated and output to ProEngineer CAD system for simulation and visualization purposes. The machined part then was simulated by subtracting the swept volume of the machining tool from the work piece. Machined surface error prediction, which is considered to be one of the most important verification objectives, was carried out by indicating the patches of the machined surface where the deformation exceeds the surface error tolerance.

By checking the machining collision and predicting the machined surface error, the manufacturers can pre-check the NC sequence in a more accurate and sufficient way and decide whether they need to modify the manufacturing setup parameters before machining. This can save manufacturing costs and shorten manufacturing time.

## 6.2 Suggestions for Future Work

As discussed before, the SDE approach is capable of integrating with more complex end milling models, although some difficulties such as computation complexity and the size of data (grazing points) will arise. The suggested tasks for future research are:

1. Extend the existing SEDE method to include general spatial deformation.
2. Develop a program to generate the deformed swept volume by the SEDE method.
3. Develop a tool deformation predictor using more complicated advanced cutting force models.

# APPENDIX A

## PROGRAM FOR CL DATA EXTRACTION

```
/*********************************************************************
  This is a program in C++ for getting the CL data from the cl data file *.ncl

  The CL data file is from Pro/Manufacturing

  It's OK for both 3-axis and 5 axis machining CL data file; Cl data which are not
  for the cutting (contacted with workpiece, such as rapid positioning, feeding) are
  not readin.

  output: cl.dat      for cl data;          mcd.dat      for mcd  checking
*********************************************************************/
# include <math.h>
# include <iostream.h>
# include <fstream.h>
# include <assert.h>
# include <stdlib.h>
# include  <string.h>
# include  <new.h>

int main()
{
 float xx0,yy0,zz0,ii0,jj0,kk0;
 float xx1,yy1,zz1,ii1,jj1,kk1;
 float x0,y0,z0,a0,b0;
 float x1,y1,z1,a1,b1;
 float x,y,z,ii,jj,kk;
 int i, j, k, current_char;
 float  amp;
 char*  line=new char[80];
 char*  string=new char[12];
/*****************************************************************
   Read in CL data block by block
*****************************************************************/
 char  Type;
 char  CL[10];
 cout<<" Please  input  CLdata file name:    ";
 cin>>CL;
 cin.clear();
 ifstream  Input_cl(CL, ios::in);
 if(!Input_cl)  {cout<<"Can not open for read in CL data";  exit(-1);}

type:  cout<<"3 axis maching or 5 axis maching ?  (3/5):    ";
```

```
cin>>Type;
if (Type!='3'&& Type!='5')  {cout<<" Wrong input !!"<<endl; goto type;}

ofstream  Output_cl("cl.dat", ios::out);
ofstream  Output_mcd("mcd.dat", ios::out);
cout<<" Amplified by : ";
cin>>amp;
read: Input_cl.getline(line, 80, '\n');
    if (line[0]!='F' || line[1]!='E' || line[2]!='D')  {goto read;}

    else {
        while (!Input_cl.eof())
           {
            Input_cl.getline(line, 80, '\n');

            if (line[0]=='G'&&line[1]=='O'&&line[2]=='T'&&line[3]=='O')
                  {
               /***********************************
                  Read in x , y, z line by line
               ***********************************/
                /************For x ****************/
                 j=0;
                 current_char=7;

                 while (line[current_char]!=',')
                  {
                   string[j]=line[current_char];
                   current_char=current_char+1; j=j+1;
                  }

                 x=atof(string);                    //string to float number
                 cout<<x<<"  ";
                 for (k=0; k<j; ++k)                  // do not output ,
                  {Output_cl<<string[k];}
                 Output_cl<<"  ";

               /***************** For y ******************/
                j=0;
                current_char=current_char+2;

               while (line[current_char]!=',')
               {string[j]=line[current_char]; current_char=current_char+1; j=j+1;}

               y=atof(string);   cout<<y<<"  ";       //string to float

               for (k=0; k<j; ++k)
               {Output_cl<<string[k];}
               Output_cl<<"  ";
```

```
/****************** For z ***********/
 j=0;
 current_char=current_char+2;

 if (Type=='3')                    // 3-axis or 5-axis ???
  {
   while (line[current_char]!=NULL)
     {string[j]=line[current_char]; current_char=current_char+1; j=j+1;}

    z=atof(string);    cout<<z<<endl;


   for (k=0; k<j; ++k)
     {Output_cl<<string[k];}
   Output_cl<<endl;
  }
/*******************5 axis cl data******************/
else {
    while (line[current_char]!=',')
       {string[j]=line[current_char]; current_char=current_char+1; j=j+1;}

    z=atof(string);    cout<<z<<endl;


   for (k=0; k<j; ++k)
      {Output_cl<<string[k];}
   Output_cl<<"    ";

   Input_cl.getline(line, 80, '\n');
 /************** For i **********************/
  j=0;
  current_char=0;

  while (line[current_char]!=',')
    {
     string[j]=line[current_char];
     current_char=current_char+1; j=j+1;
    }

 ii=atof(string);                  //string to float number
 cout<<ii<<"   ";

 for (k=0; k<j; ++k)               // do not output' ,'
   {Output_cl<<string[k];}
 Output_cl<<"   ";
/****************For j ******************/
 j=0;
 current_char=current_char+2;
```

```
                while (line[current_char]!=',')
                   {
                     string[j]=line[current_char];
                     current_char=current_char+1; j=j+1;
                   }

                jj=atof(string);                    //string to float number
                cout<<jj<<"   ";

                for (k=0; k<j; ++k)                  // do not output' ,'
                   {Output_cl<<string[k];}
                Output_cl<<"   ";
               /************ For k ************/
                j=0;
                current_char=current_char+2;

                while (line[current_char]!=NULL)
                   {
                     string[j]=line[current_char];
                     current_char=current_char+1; j=j+1;
                   }

                kk=atof(string);                    //string to float number
                cout<<kk<<"   ";

                for (k=0; k<j; ++k)                  // do not output' ,'
                   {Output_cl<<string[k];}
                Output_cl<<endl;
            }                       //else for 5 axis
         }            // if
      }               //while

   }
        Input_cl.close(); Output_cl.close();
        delete[] line; delete[] string;
        exit(-1);
}
```

# APPENDIX B

## PROGRAM FOR DEFORMED SWEPT VOLUMES GENERATION

```
/************************************************************
                    This program is developed by
                              Feng Lu
                  Robotics & Intelligent Manufacturing Lab.
                                NJIT

                          Last modified in Aug. 1997
 ************************************************************/
/*****************************************************************
This program was developed in C++;

It has the capability of reading_ in multi_block CL data and generate the whole deformed swept
volume according to the all CL data. Thus it will save a lot of time for the visualization and
cutout.

This program is for continuous block swept volumes generation; Flat and Ball_end tool

1. Read in CL data block by block;  Tool parameter; Cutting forces data;
  (Also capable of build_in cutting force prediction, deformation type3)

2. Creating the deformation and deflection of
  the milling tool in mill process

3. SDE with deformation;

4. Output data for Pro-E readable file *.ibl

5. Output point data for verification; Output the cutting force as well
  as deformation/deflection
*****************************************************************/
# include <math.h>
# include <iostream.h>
# include <fstream.h>
# include <assert.h>
# include  <stdlib.h>

/*****************************************************
   Simulation Parameter and Constants Definition
*****************************************************/
# define  Pi  3.1415926
# define  E   200.0e3
# define  R   3.175            //  Units is : mm
```

```
# define  L   38.08
# define  Eh  0.03          //deflection coefficient
# define  Er  13.0          //deflection coefficient

# define  Psp     1.71e3
# define  Spindle 500             // (rpm)
# define  Feed    0.02            // (mm/tooth)
# define  Radius_depth   6.35           // radial depth of cut, mm
# define  Flute   4             // number of flute

# define  Z_Div 40            //division along z axis
# define  Amp  1
/* # define  TT  100    */       //time division
# define  Minimum  0.001
# define  Theta_div  36       //Theta divisions
# define  Tolerance  0.013
# define  Depth_cut  20.0       //depth of cut: 5.0 mm

/*********************************
 Data Structure Definition
*********************************/
typedef  struct  {
           int row,col;
           float ele[3][3];
          } Matrix;

typedef  struct  { float x, y, z; } point;

typedef  struct {                  // define a section struct
                          //  for grazing points in one section
           int nums;
           point pt[2];
          } Section;

/************************************************************
  Function Definition
  Note: All the calculation are in Matrix form 3x3 or 3x1
************************************************************/
extern Matrix   Matrix_add(Matrix, Matrix);
extern Matrix   Matrix_mult1(Matrix, Matrix);
extern double   Matrix_dotmult(Matrix, Matrix);
extern Matrix   Differential(int, int, Matrix, Matrix, float);
extern void    output(double, int, int, int, float, int);
extern Matrix   Pmatrix(float);
extern Matrix   Pdotmatrix(float);
extern Matrix   Tmatrix(float);
extern Matrix   Tdotmatrix(float);

/*********************************
```

```
  Global variable
************************************/
int Div;
float LL;
int TT;
char  Optimize,Finish_YN;

int block;          // for multiple block

float Theta_test[100];


double Theta_t0[Z_Div][2], Theta_t1[Z_Div][2];

float Dispx[400][100], Dispy[400][100], Dispx_pre[400][100], Dispy_pre[400][100],
Dispx_next[400][100], Dispy_next[400][100];
float xx0,yy0,zz0,ii0,jj0,kk0;
float xx1,yy1,zz1,ii1,jj1,kk1;

float XX[20],YY[20],ZZ[20],II[20],JJ[20],KK[20];          //for multiblock!!!

Matrix P;                    /* P.row=3;     P.col=1; */
Matrix P_dot;                  /* P_dot.row=3;    P_dot.col=1; */
Matrix T;                   /* T.row=3;    T.col=3; */
Matrix T_dot;                  /* T_dot.row=3;  T_dot.col=3;  */

Matrix Deformation, Deform_dot;
Section Grazing_slice[200][100];

ofstream checking("check.dat", ios::out);
ofstream  outgrazing("grazing.ibl", ios::out);
ofstream  outpoint("verpoints.pts", ios::out);
ofstream  outpoints("grazing.pts", ios::out);
ofstream  exceed1("tolerance.ibl", ios::out);
ofstream outfile2("force.out", ios::out);
ofstream outfile1("deform.out", ios::out);
fstream  maxdeform_out("Maxdeform.out", ios::out | ios::in);

/********************* Matrix Operations Functions *************************/

/******************** mxn matrix  *   nxk matrix ************/
 Matrix  Matrix_mult1 (int m, int n, int k, Matrix M1, Matrix M2)
 {
  Matrix   N;

  int i, j, w;
  M1.row=m;   M1.col=n;
  M2.row=n;   M2.col=k;
  N.row=m;    N.col=k;
```

```
  for (w=0; w<=k-1; ++w)
      {
       for (i=0; i<=m-1; ++i)
          {
            N.ele[i][w]=0;
            for (j=0;j<=n-1;++j)
              {
               N.ele[i][w]=N.ele[i][w]+M1.ele[i][j]*M2.ele[j][w];
              }
          }
      }
  return N;
 }

/************************ matrix addition ********************/
Matrix  Matrix_add (int m,  int n,  Matrix M1, Matrix M2)
 {
   Matrix N;
   int i,j;
   M1.row=m;   M1.col=n;
   M1.row=m;   M2.col=n;
   N.row=m;    N.col=n;

   for (i=0; i<=m-1; ++i)
     {
      for (j=0; j<=n-1; ++j)
        {
         N.ele[i][j]=M1.ele[i][j]+M2.ele[i][j];
        }
     }
   return N;
}

Matrix Differential (int m, int n,  Matrix D1, Matrix D2, float DetaT)
 {
  Matrix N;
  int i,j;
  D1.row=m;   D1.col=n;
  D2.row=m;   D2.col=n;
  N.row=m;   N.col=n;
  for (i=0; i<=m-1; ++i)
    {
     for (j=0; j<=n-1; ++j)
       {
        N.ele[i][j]=(D2.ele[i][j]-D1.ele[i][j])/DetaT;
       }
    }
  return N;
 }
```

```
/************************ vector dotmultiply ********************/
double Matrix_dotmult(Matrix M1, Matrix M2)
{
  double   Dotmult;
  Dotmult=M1.ele[0][0]*M2.ele[0][0]+M1.ele[1][0]*M2.ele[1][0]+M1.ele[2][0]*M2.ele[2][0];
  return   Dotmult;

}

/************************ Matrix  Operations  End ************************/


/*************** Motion Equation definition ******************************
 Read in CL data, create the transform matrix P, T, P_dot, T_dot
*************************************************************************/
/*****************************************************
   CL data  to   Motion
*****************************************************/

Matrix Pmatrix(float t)
{
 Matrix trans;
 trans.row=3;  trans.col=1;

 trans.ele[0][0]=xx0+(xx1-xx0)*t;
 trans.ele[1][0]=yy0+(yy1-yy0)*t;
 trans.ele[2][0]=zz0+(zz1-zz0)*t;
 return trans;
}

Matrix Pdotmatrix(float t)
{
 Matrix trans;
 trans.row=3;  trans.col=1;

 trans.ele[0][0]=  xx1-xx0;
 trans.ele[1][0]=  yy1-yy0;
 trans.ele[2][0]=  zz1-zz0;
 return trans;
}


Matrix Tmatrix(float t)
{
 Matrix trans;
 trans.row=3;  trans.col=3;

 float it, jt, kt;
 float ca,sa,cb,sb;
```

```
 float a0, b0, a1, b1, a, b;

 a0=asin(jj0);
 if (fabs(ii0)<=0.00001)  {b0=0.0;}
   else  {if (kk0!=0) {b0=-atan(ii0/kk0);}   else {b0=Pi/2.0;}}

 a1=asin(jj1);
 if (fabs(ii1)<=0.00001)  {b1=0.0;}
   else  {if (kk1!=0) {b1=-atan(ii1/kk1);}   else {b1=Pi/2.0;}}


 a=a0+(a1-a0)*t;
 b=b0+(b1-b0)*t;

 ca=cos(a); sa=sin(a); cb=cos(b); sb=sin(b);

 trans.ele[0][0]=cb; trans.ele[0][1]=sa*sb; trans.ele[0][2]=-ca*sb;
 trans.ele[1][0]=0; trans.ele[1][1]=ca;  trans.ele[1][2]=sa;
 trans.ele[2][0]=sb;   trans.ele[2][1]=-sa*cb;   trans.ele[2][2]=ca*cb;

 return trans;
}


Matrix Tdotmatrix(float t)

{
 Matrix trans;
 trans.row=3;  trans.col=3;

 float a0, b0, a1, b1, a, b, a_dot,b_dot;
 float ca,sa,cb,sb;

 a0=asin(jj0);
 if (fabs(ii0)<=0.00001)  {b0=0;}
   else  {if (kk0!=0) {b0=-atan(ii0/kk0);} else {b0=Pi/2.0;}}

 a1=asin(jj1);
 if (fabs(ii1)<=0.00001)  {b1=0;}
    else  {if (kk1!=0) {b1=-atan(ii1/kk1);} else {b1=Pi/2.0;}}

 a=a0+(a1-a0)*t;
 b=b0+(b1-b0)*t;

 a_dot=a1-a0;
 b_dot=b1-b0;

 ca=cos(a); sa=sin(a); cb=cos(b); sb=sin(b);
```

```
 trans.ele[0][0]=-sb*b_dot;    trans.ele[0][1]=ca*a_dot*sb+sa*cb*b_dot;
trans.ele[0][2]=sa*sb*a_dot-ca*cb*b_dot;
 trans.ele[1][0]=0;          trans.ele[1][1]=-sa*a_dot;            trans.ele[1][2]=ca*a_dot;
 trans.ele[2][0]=cb*b_dot;   trans.ele[2][1]=-ca*cb*a_dot+sa*sb*b_dot;  trans.ele[2][2]=-
sa*cb*a_dot-ca*sb*b_dot;

 return trans;
}

/*************** Motion end ***********************/



/******************** Deformation Calculations ***********************/

/************** None deform *********************/
void nondeform()
{
 TT=10;
 Optimize='n';
 int i, j;
 for (i=0; i<=TT; ++i)
   {
    for(j=0; j<=Z_Div; ++j)
      {
        Dispx[i][j]=0.0;
        Dispy[i][j]=0.0;
      }
   }
}


/************* deformation case 1 ******************
OK !!!!     Keyboard input averaging cutting force
*******************************************/
void deform1()
  {
      float  Fx, Fy;
      float DLx1[200], DLy1[200], DFx1[200], DFy1[200];
      float  D,J, dz;
      int   i, j;

      float  Max_deform, Deform_slice;

      TT=10;
      D=2*R;   J=Pi*D*D*D*D/48;
      dz=L/Z_Div;

      cout<<"Please input the average cutting forces in X,Y direction"<<endl<<"Fx=    ";
```

```
        cin>>Fx;
        cout<<"Fy=    ";
        cin>>Fy;

        ofstream  outfile1("deform", ios::out);
        outfile1<<"This is the output file of the displacement in case 1"<<endl;
        outfile1<<"Time Instance    Z-position      Displacement in X     Displacement in
Y"<<endl;


        for (i=0; i<=Z_Div; ++i)
          {
            DLx1[i]= (Eh*Fx+Er*Fx*L/1000*(dz*i)/1000)/1000;
            DLy1[i]= (Eh*Fy+Er*Fy*L/1000*(dz*i)/1000)/1000;      // mm


            DFx1[i]= (Fx*(dz*i)*(dz*i)*(3*L-dz*i)/(6*E*J));
            DFy1[i]= (Fy*(dz*i)*(dz*i)*(3*L-dz*i)/(6*E*J));     // mm

          }
       /******* amplify the deformation by 200 times**********************/
         for (i=0; i<=TT; ++i)
           {

            Max_deform=0.0;

            for (j=0; j<=Z_Div; ++j)
              {
                Dispx[i][j]=-(DLx1[j]+DFx1[j])*Amp;
                Dispy[i][j]=-(DLy1[j]+DFy1[j])*Amp;

                Deform_slice=sqrt(Dispx[i][j]*Dispx[i][j]+Dispy[i][j]*Dispy[i][j]);
                if (Max_deform<Deform_slice)  {Max_deform=Deform_slice;}


                outfile1<<i<<"          "<<L/Z_Div*j<<"          "<<Dispx[i][j]<<"
"<<Dispy[i][j]<<endl;

              }
            outfile1<<"          Maxium Deformation in this time instance:
"<<Max_deform<<"\n"<<endl;
           }
       TT=30;
    }


/************** deformation case 2 ******************
OK !!!!!!!!!
In this case we read in the simulated cutting force from Dr. Devor's software;
```

```
According to finish/rough cut, we can optimize cutting forces by only read in
maximum/minimum cutting force or average force in each rotation
*****************************************************/
int deform2_yes()
   {
      int i,j, Time_div;
      float Theta1,Theta2,Theta3, First;
      char No[10];
      float  Fx1, Fy1, Fx2, Fy2, Fx3, Fy3, Total_x, Total_y;
      float Fx[400], Fy[400], Fx_pre[400], Fy_pre[400], Fx_next[400], Fy_next[400],
Theta_rec[400];
      float DLx2, DLy2, DFx2, DFy2;


      float  D,J,dz;

      float  Max_total, Max_deform, Deform_slice;


      D=2*R;   J=Pi*D*D*D*D/48;
      dz=L/Z_Div;

       char  force[10];

       cout<<"Please input the cutting force data file name:  ";
       cin>>force;
       ifstream infile(force, ios::in);
       if (!infile)
         {
          cerr<<"cannot open file for input\n";
          exit(-1);
         }
    /*************************** read in the force data ********************
                        Change unit: lbf ----- N
     ***********************************************************************/
        infile>>First;

    cout<<"Finish Cut ? (y/n):  ";
    cin>>Finish_YN;

  /************************ Finish Cut Approximation ********************/
    if (Finish_YN=='Y' || Finish_YN=='y')
      {

       Fx_pre[0]=0;  Fy_pre[0]=0; Fx[0]=0; Fy[0]=0; Fx_next[0]=0; Fy_next[0]=0;

       i=1;

       infile>>Theta1>>Fx1>>Fy1>>First>>First;
```

```
        infile>>Theta2>>Fx2>>Fy2>>First>>First;

        while (infile.eof()==0)
         {
           infile>>Theta3>>Fx3>>Fy3>>First>>First;

           if ((Fy2>Fy1 && Fy2>Fy3) || ((Fy2<=Fy1 && Fy2<=Fy3)&&(Fy1+Fy2+Fy3!=0)))
             {
               Fx_pre[i]=Fx1*4.448/4.448;    Fy_pre[i]=Fy1*4.448/4.448;
               Fx[i]=Fx2*4.448/4.448;        Fy[i]=Fy2*4.448/4.448;
               Fx_next[i]=Fx3*4.448/4.448;   Fy_next[i]=Fy3*4.448/4.448;
               Theta_rec[i]=Theta2;
               ++i;
             }

           Fx1=Fx2; Fy1=Fy2;  Fx2=Fx3; Fy2=Fy3;
           Theta1=Theta2;  Theta2=Theta3;
         }

       Time_div=i-1;
        cout<<"    Number of Forces read:  "<<Time_div<<endl;

        outfile2<<"This is the output file of the displacement in case 2"<<endl;
        outfile2<<"Total Optimized Forces Read in:   "<<Time_div<<endl;
        outfile2<<"Rotational Deg.  Force_X  Force_Y    Force_previous_x   Force_previous_y
Force_next_x   Force_next_y"<<endl;
        for (i=0; i<=Time_div; ++i)
          {
           outfile2<<Theta_rec[i]<<"          "<<Fx[i]<<"        "<<Fy[i]<<"
"<<Fx_pre[i]<<"          "<<Fy_pre[i]<<"          "<<Fx_next[i]<<"     "<<Fy_next[i]<<endl;
          }
        outfile2.close();

    }

  /********************** Rough Cut Approximation **********************/

   else
    {
      i=0;
      while (infile.eof()==0)
       {
        Total_x=0.0; Total_y=0.0;
        for (j=0; j<=71; ++j)
          {
           infile>>Theta1>>Fx1>>Fy1>>First>>First;
           Total_x=Total_x+Fx1;
           Total_y=Total_y+Fy1;
          }
```

```
        Fx[i]=Total_x/72;
        Fy[i]=Total_y/72;
        ++i;
       }

      Time_div=i-1;

      outfile2<<"This is the output file of the displacement in case 2"<<endl;
      outfile2<<"Total Average Forces :  "<<Time_div<<endl;
      outfile2<<"Rotational  Force_X      Force_Y    "<<endl;
      for (i=0; i<=Time_div; ++i)
         {
          outfile2<<i<<"          "<<Fx[i]<<"        "<<Fy[i]<<endl;
         }

   }

       outfile1<<"This is the output file of the displacement in case 1"<<endl;
       outfile1<<"Time_instance   Z-position      Displacement in X     Displacement in
Y"<<endl;

       Max_total=0.0;
       for (i=0; i<=Time_div; ++i)
        {

          Max_deform=0.0;

          for (j=0; j<=Z_Div; ++j)
             {
              DLx2= (Eh*Fx_pre[i]+Er*Fx_pre[i]*L/1000*(L-dz*j)/1000)/1000;
              DLy2= (Eh*Fy_pre[i]+Er*Fy_pre[i]*L/1000*(L-dz*j)/1000)/1000;

              DFx2= Fx_pre[i]*(L-dz*j)*(L-dz*j)*(2*L+dz*j)/(6*E*J);
              DFy2= Fy_pre[i]*(L-dz*j)*(L-dz*j)*(2*L+dz*j)/(6*E*J);

              Dispx_pre[i][j]=(DLx2+DFx2)*Amp;
              Dispy_pre[i][j]=(DLy2+DFy2)*Amp;



              DLx2= (Eh*Fx[i]+Er*Fx[i]*L/1000*(L-dz*j)/1000)/1000;
              DLy2= (Eh*Fy[i]+Er*Fy[i]*L/1000*(L-dz*j)/1000)/1000;

              DFx2= Fx[i]*(L-dz*j)*(L-dz*j)*(2*L+dz*j)/(6*E*J);
              DFy2= Fy[i]*(L-dz*j)*(L-dz*j)*(2*L+dz*j)/(6*E*J);

              Dispx[i][j]=-(DLx2+DFx2)*Amp;
              Dispy[i][j]=-(DLy2+DFy2)*Amp;
```

```
            DLx2= (Eh*Fx_next[i]+Er*Fx_next[i]*L/1000*(L-dz*j)/1000)/1000;
            DLy2= (Eh*Fy_next[i]+Er*Fy_next[i]*L/1000*(L-dz*j)/1000)/1000;

            DFx2= Fx_next[i]*(L-dz*j)*(L-dz*j)*(2*L+dz*j)/(6*E*J);
            DFy2= Fy_next[i]*(L-dz*j)*(L-dz*j)*(2*L+dz*j)/(6*E*J);

            Dispx_next[i][j]=-(DLx2+DFx2)*Amp;
            Dispy_next[i][j]=-(DLy2+DFy2)*Amp;

            Deform_slice=sqrt(Dispx[i][j]*Dispx[i][j]+Dispy[i][j]*Dispy[i][j]);
            if (Max_deform<Deform_slice)  {Max_deform=Deform_slice;}

            outfile1<<i<<"          "<<L/Z_Div*j<<"          "<<Dispx[i][j]<<"
"<<Dispy[i][j]<<endl;

          }
        outfile1<<"          Maxium Deformation in this time instance:
"<<Max_deform<<"\n"<<endl;
        if (Max_total<Max_deform)  {Max_total=Max_deform;}
       }

     outfile1<<endl;
     outfile1<<"          Maxium Deformation in all Simulation:     "<<Max_total<<endl;
     TT=Time_div;
     return Time_div;
  }


/*************************************************************
 Almost the same as deform2_yes except we do not optimize cutting forces.
*************************************************************/
int deform2_no()
{
     int i,j, Time_div;
     float Theta_rec[100], First;
     char No[10];
     float Max_deform, Deform_t;
     float Fx2[100], Fy2[100];
     float DLx2[100], DLy2[100], DFx2[100], DFy2[100];
     float  D,J,dz;

     D=2*R;   J=Pi*D*D*D*D/48;
     dz=L/Z_Div;

      char  force[10];

      cout<<"Please input the cutting force data file name:  ";
```

```
        cin>>force;
        ifstream infile(force, ios::in);
        if (!infile)
          {
           cerr<<"cannot open file for input\n";
           exit(-1);
          }

         infile>>First;

         i=0;
         while (infile.eof()==0)
          {  infile>>Theta_rec[i]>>Fx2[i]>>Fy2[i]>>First>>First;
             Fx2[i]=Fx2[i]*4.448/4.448;  Fy2[i]=Fy2[i]*4.448/4.448;

             ++i;
          }



        Time_div=i-2;
        cout<<"    Number of Forces read:  "<<Time_div<<endl;

        outfile2<<"This is the output file of the displacement in case 2"<<endl;
        outfile2<<"Total Forces Read in:   "<<Time_div<<endl;
        outfile2<<"Rotational Deg.  Force_X  Force_Y   "<<endl;
        for (i=0; i<=Time_div; ++i)
          {
           outfile2<<Theta_rec[i]<<"          "<<Fx2[i]<<"         "<<Fy2[i]<<endl;
          }

        outfile1<<"This is the output file of the displacement in case 2"<<endl;
        outfile1<<"Total Forces Read in:     "<<Time_div<<endl;
        outfile1<<"Time_instance   Z-position      Displacement in X     Displacement in
Y"<<endl;

        for (i=0; i<=Time_div; ++i)
         {
          Max_deform=0.0;
          outfile1<<endl;
          for (j=0; j<=Z_Div; ++j)
            {
              DLx2[j]= (Eh*Fx2[i]+Er*Fx2[i]*L/1000*(L-dz*j)/1000)/1000;
              DLy2[j]= (Eh*Fy2[i]+Er*Fy2[i]*L/1000*(L-dz*j)/1000)/1000;

              DFx2[j]= Fx2[i]*(L-dz*j)*(L-dz*j)*(2*L+dz*j)/(6*E*J);
              DFy2[j]= Fy2[i]*(L-dz*j)*(L-dz*j)*(2*L+dz*j)/(6*E*J);

              Dispx[i][j]=-(DLx2[j]+DFx2[j])*Amp;
```

```
            Dispy[i][j]=-(DLy2[j]+DFy2[j])*Amp;

            Deform_t=sqrt(Dispx[i][j]*Dispx[i][j]+Dispy[i][j]*Dispy[i][j]);
            if (Max_deform<Deform_t)  {Max_deform=Deform_t;}                    //record
maximum deform in time t

            outfile1<<i<<"          "<<L/Z_Div*j<<"          "<<Dispx[i][j]<<"
"<<Dispy[i][j]<<endl;


          }
          outfile1<<"          Maximum Deformation in this time section:
"<<Max_deform<<endl;
          outfile1<<endl;
        }

      TT=Time_div;

      return Time_div;
  }


/*************************************************************************
   Build_in Average Cutting Force Model
*************************************************************************/
void deform3()
{
      int i,j, Time_div, Milltype;

      TT=20;

      char Type, Multi_YN;
      char *Previous_cut=new char[10];

      float MRR, Axial_depth;
      float Fx[400], Fy[400];
      float DLx2, DLy2, DFx2, DFy2;
      float  D,J,dz;
      float Extra_cross, Readin_deform;
      float  Max_total, Max_deform, Deform_slice;
      ifstream Pre_deform;


      D=2*R;   J=Pi*D*D*D*D/48;
      dz=L/Z_Div;

type:    cout<<" Build in cutting force prediction : Milling type  (U)p / (D)own ?"<<endl;
      cin>>Type;
      if (Type=='U'||Type=='u')  {Milltype=1;}
```

```
        else if (Type=='D'||Type=='d') {Milltype=-1;}
        else  {cout<<"Wrong input !!"; goto type;}

        cout<<"Multi_pass simulation ? (Y/N):  ";
        cin>>Multi_YN;
        i=0;
        if (Multi_YN=='Y' || Multi_YN=='y')
                  {  cout<<"Which previous deformed swept volume as feedback ? : ";
                     cin>>Previous_cut;
                     Pre_deform.open(Previous_cut, ios::in);
                     if (!Pre_deform) {cerr<<"No previous cut !"; exit(-1);}
                  }

      /************************** Calculate the force data ********************
                        Change unit: lbf ----- N     no
       ***************************************************************/
        for (i=0; i<=TT;++i)
            {
              Axial_depth=fabs(zz0+(zz1-zz0)*i/TT);
              if (Multi_YN=='Y' || Multi_YN=='y')
                 {
                  Pre_deform>>Readin_deform;
                  Extra_cross=Readin_deform*Axial_depth;
                  MRR=(Radius_depth*Axial_depth+Extra_cross)*Flute*Feed*Spindle;
                 }
              else { MRR=Radius_depth*Axial_depth*Flute*Feed*Spindle;}
              Fy[i]=Psp*MRR/(Pi*D*Spindle)*Milltype;
              Fx[i]=-fabs(Fy[i]/2);
            }

        outfile2<<"This is the output file of the cutting force in case 3"<<endl;

        outfile2<<"Section    Force_X      Force_Y   "<<endl;
        for (i=0; i<=TT; ++i)
          {
           outfile2<<i<<"          "<<Fx[i]<<"        "<<Fy[i]<<"          "<<endl;
          }

        outfile1<<"This is the output file of the displacement in case 3"<<endl;
        outfile1<<"Time_instance    Z-position      Displacement in X      Displacement in
Y"<<endl;
        Max_total=0.0;
        for (i=0; i<=TT; ++i)
         {
           Max_deform=0.0;

           for (j=0; j<=Z_Div; ++j)
               {
                DLx2= (Eh*Fx[i]+Er*Fx[i]*L/1000*(L-dz*j)/1000)/1000;
```

```
            DLy2= (Eh*Fy[i]+Er*Fy[i]*L/1000*(L-dz*j)/1000)/1000;

            DFx2= Fx[i]*(L-dz*j)*(L-dz*j)*(2*L+dz*j)/(6*E*J);
            DFy2= Fy[i]*(L-dz*j)*(L-dz*j)*(2*L+dz*j)/(6*E*J);

            Dispx[i][j]=-(DLx2+DFx2)*Amp;
            Dispy[i][j]=-(DLy2+DFy2)*Amp;

            Deform_slice=sqrt(Dispx[i][j]*Dispx[i][j]+Dispy[i][j]*Dispy[i][j]);
            if (Max_deform<Deform_slice) {Max_deform=Deform_slice;}

            outfile1<<i<<"        "<<L/Z_Div*j<<"        "<<Dispx[i][j]<<"
"<<Dispy[i][j]<<endl;

          }
        outfile1<<"        Maxium Deformation in this time instance:
"<<Max_deform<<"\n"<<endl;

        maxdeform_out<<Max_deform<<"    "<<endl;

        if (Max_total<Max_deform) {Max_total=Max_deform;}
      }

    outfile1<<endl;
    outfile1<<"        Maxium Deformation in all Simulation:    "<<Max_total<<endl;
}


/*************************************************************
  The calling function for deformation calculation
*************************************************************/
void Deform_calculation()
{
 char None;
 int model, tt;
 char names[10];
 deform:  cout<<"Do you want the Deformation? (Y/N)\n ";
 cin>>None;
 cin.clear();
 if (None=='N' || None=='n')
   { nondeform();}
 else
   {
     if (None=='Y' || None=='y')
       {
         repeat:  cout<<"Please select the cutting force model: model 1, 2, or 3"<<endl;
         cin>>model;
         cin.clear();
         switch (model)
```

```
            {
              case 1:
                   deform1();
                   break;

              case 2:
                   {
                    cout<<"Want Optimize Cutting Force Input ?  (Y/N):  ";
                    cin>>Optimize;
                    if (Optimize=='N' || Optimize=='n')
                      {
                         tt=deform2_no();
                      }
                    else  if (Optimize=='Y' || Optimize=='y')
                         {
                          tt=deform2_yes();
                         }
                    else  {cout<<"Wrong select !!!!!!"<<endl;  exit(1);}

                    cout<<"                    Time:"<<tt<<endl;
                    break;
                   }
              case 3:
                   deform3();
                   break;

              default:
                   cout<<"wrong input"<<endl;
                   goto repeat;
                   break;
            }
          }
        else
          { cout<<"Wrong Input !!!\n";
           goto deform;}
      }

  cout<<" DONE FORCE/DEFORMATION IN THIS BLOCK  !"<<endl;

}

/********************* Deformation  Ends ************************/


/****************** Transform tool coord. to global coord. sys. **********/

Matrix  Transform( Matrix Deformationt, point Localt)
{
 Matrix  Rotate, RR, Deformed;
```

```
 Rotate.row=3;  Rotate.col=1;
 RR.row=3;     RR.col=1;
 Deformed.row=3;  Deformed.col=1;

 Deformed.ele[0][0]=Localt.x+Deformationt.ele[0][0];
 Deformed.ele[1][0]=Localt.y+Deformationt.ele[1][0];
 Deformed.ele[2][0]=Localt.z+Deformationt.ele[2][0];

 Rotate=Matrix_mult1(3,3,1,T,Deformed);

 RR=Matrix_add(3,1,P,Rotate);

 return RR;
}



/************************************************************
 Organize the grazing points slice by slice; interval by interval
************************************************************/
void output(double theta, int timeth, int sliceth, int point_number, float r, int type)

{
  point A_local, A_global;
  Matrix A;       A.row=3;     A.col=1;

  if (timeth==0)  {Theta_t0[sliceth][point_number]=theta;}
  else if (timeth==TT)  {Theta_t1[sliceth][point_number]=theta;}

  if (type==1)
     {
       A_local.x=r*cos(theta);
       A_local.y=r*sin(theta);
       A_local.z=LL/Div*sliceth;
     }
  else if (type==2)
     {

       A_local.x=r*cos(theta);
       A_local.y=r*sin(theta);
       A_local.z=R/Div*sliceth;
     }
  else if (type==3)
     {
       A_local.x=r*cos(theta);
       A_local.y=r*sin(theta);
       A_local.z=R+(LL-R)/Div*(sliceth-Div);
     }
```

```
  A=Transform(Deformation, A_local);

  checking<<"                    "<<A.ele[0][0]<<"    "<<A.ele[1][0]<<"    "<<A.ele[2][0]<<
endl;

  A_global.x=A.ele[0][0];
  A_global.y=A.ele[1][0];
  A_global.z=A.ele[2][0];

   /********* save the grazing point in Grazing section*********/

  Grazing_slice[timeth][sliceth].pt[point_number]=A_global;
}


/******************** calculate   SDE ********************************/

Matrix SDE(Matrix Deformations, point Locals, Matrix Deform_dots)

{
  Matrix inter;     inter.row=3;  inter.col=1;
  Matrix R_dots;     R_dots.row=3;  R_dots.col=1;
  Matrix Deformed;   Deformed.row=3;  Deformed.col=1;
  Matrix  mult1, mult2;
  mult1.row=3;    mult1.col=1;
  mult2.row=3;    mult2.col=1;

 Deformed.ele[0][0]=Locals.x+Deformations.ele[0][0];
 Deformed.ele[1][0]=Locals.y+Deformations.ele[1][0];
 Deformed.ele[2][0]=Locals.z+Deformations.ele[2][0];

  mult1=Matrix_mult1(3,3,1,T_dot, Deformed);
  mult2=Matrix_mult1(3,3,1,T, Deform_dots);

  inter=Matrix_add(3,1,P_dot, mult1);

  R_dots=Matrix_add(3,1,inter,mult2);

  return  R_dots;

}

/*************************************************************
 calculate Tangency funcation at each point      OK!!
*************************************************************/
double Tangency_function(double Theta, int z_div, float r, int type)
{
   double tangency;
   Matrix N;      N.row=3;         N.col=1;
```

```
  Matrix R_dot;  R_dot.row=3;     R_dot.col=1;
  Matrix N_bar;  N_bar.row=3;     N_bar.col=1;
  point a_local;

  if (type==1)
    {
        a_local.x=r*cos(Theta);
        a_local.y=r*sin(Theta);
        a_local.z=LL/Div*z_div;

/************** Outter vector definition *****************/
        N.ele[0][0]=r*cos(Theta);
        N.ele[1][0]=r*sin(Theta);                    //OK!
        N.ele[2][0]=0;
    }

  else if (type==2)
    {

        a_local.x=r*cos(Theta);
        a_local.y=r*sin(Theta);
        a_local.z=R/Div*z_div;

/************** Outter vector definition *****************/

        N.ele[0][0]=r*cos(Theta);
        N.ele[1][0]=r*sin(Theta);                    //OK!
        N.ele[2][0]=R/(Div)*z_div-R;

    }
  else if (type==3)
    {
        a_local.x=r*cos(Theta);
        a_local.y=r*sin(Theta);
        a_local.z=R+(LL-R)/Div*(z_div-Div);

/************** Outter vector definition *****************/

        N.ele[0][0]=r*cos(Theta);
        N.ele[1][0]=r*sin(Theta);                    //OK!
        N.ele[2][0]=0;
    }

  N_bar=Matrix_mult1(3,3,1,T,N);              //OK!
  R_dot=SDE(Deformation,a_local,Deform_dot);
  tangency=Matrix_dotmult(R_dot,N_bar);
  return tangency;
}
```

```
/*********************************************************
  Find the grazing points in each section      (iteration)
*********************************************************/
float Grazing_section( int time, int slice, float r, int type )

{
 int i,div, flag , numberth       ;
 double Theta, Tangent, Tangent0, Dtheta,  Theta_d, Theta_rec[2], Tran;

/*****  Theta_d: degree          Theta: radius *********************/

 point  A_local;
 Dtheta=360.0/Theta_div;                        //degree

 Tangent0=Tangency_function(0,slice,r,type);

 numberth=0;

 if (fabs(Tangent0)<=Minimum)
    {                           /* record the point*/
    cout<<"";
    }
 if (Tangent0>0)  {flag=1;}
 else           {flag=-1;}

 for (i=1;i<=Theta_div; ++i)
  {
    Theta_d=Dtheta*i;
    Theta=Theta_d/180.*Pi;
    Tangent=Tangency_function(Theta, slice,r,type);

    if (fabs(Tangent)<=Minimum)
        {                           /* record the point*/

          Theta_rec[numberth]=Theta;
          ++numberth;
          flag=flag*(-1);
        }

    else if (flag*Tangent<0)
         {

/* cout<<" Begin recur !!"<<endl;  */

            div=1;
            Theta_d=Theta_d-Dtheta/(2.*div);
            Theta=Theta_d/180.*Pi;
```

```
            while(fabs(Tangency_function(Theta, slice,r,type))>Minimum)
              {
               div=div+1;
               if ((flag*Tangency_function(Theta, slice,r,type))<0)
                  { Theta_d=Theta_d-Dtheta/pow(2.,div);
                    Theta=Theta_d/180.*Pi;
                  }
               else
                  { Theta_d=Theta_d+Dtheta/pow(2.,div);
                    Theta=Theta_d/180.*Pi; }
               }

           Theta_rec[numberth]=Theta;
           ++numberth;

/* cout<<" Finished"<<endl;  */


            flag=flag*(-1);
           }
    }

  if ((time!=0 || block!=0) &&( fabs(Theta_rec[0]-Theta_test[slice])>Pi/2.0
&&fabs(Theta_rec[0]-Theta_test[slice])<Pi))
      {
       Tran=Theta_rec[0];
       Theta_rec[0]=Theta_rec[1];
       Theta_rec[1]=Tran;    cout<<"changed !"<<endl;
      }
  for (i=0; i<=numberth-1; ++i)
      { output(Theta_rec[i], time, slice,i,r,type); }

  if (fabs(Tangent0)<=Minimum)
    {                          /* record the point*/
    cout<<"";
    }

  return Theta_rec[0];

}

/*************************************************************************
          Find the grazing set for flat end tool for one block
*************************************************************************/
void  Grazingset_flat(  )
{
  int i,j                  ;

  Deformation.row=3;  Deformation.col=1;
```

```
  Deform_dot.row=3;  Deform_dot.col=1;
  point A_local;
  float DeltT=1.0/TT;
  float rr;
  Div=Z_Div;

  for (i=0; i<=TT; ++i)
    {

/****************** declare  P, P_dot, T, T_dot *********************/

      P=Pmatrix(DeltT*i);
      T=Tmatrix(DeltT*i);
      P_dot=Pdotmatrix(DeltT*i);
      T_dot=Tdotmatrix(DeltT*i);

      for (j=0; j<=Div; ++j)
         {
          checking<<" Time:   "<<i<<"  Section:   "<<j<<endl;

          Deformation.ele[0][0]=Dispx[i][j];
          Deformation.ele[1][0]=Dispy[i][j];
          Deformation.ele[2][0]=0;

          if (Finish_YN=='Y' || Finish_YN=='y')
             {
               Deform_dot.ele[0][0]=(0.5*Dispx_pre[i][j]-
2*Dispx[i][j]+1.5*Dispx_next[i][j])/(1*DeltT);
               Deform_dot.ele[1][0]=(0.5*Dispy_pre[i][j]-
2*Dispy[i][j]+1.5*Dispy_next[i][j])/(1*DeltT);
               Deform_dot.ele[2][0]=0;
             }
          else
             {
              Deform_dot.ele[0][0]=(Dispx[i+1][j]-Dispx[i][j])/DeltT;
              Deform_dot.ele[1][0]=(Dispy[i+1][j]-Dispy[i][j])/DeltT;
              Deform_dot.ele[2][0]=0;
             }

          rr=R;
          Theta_test[j]=Grazing_section(i,j,rr,1);
         }
    }
}

/*********************************************************************
          Find the grazing set for Ball_end tool for one block
*********************************************************************/
void  Grazingset_ball(  )
```

```
{
  int i,j                    ;

  Deformation.row=3;  Deformation.col=1;
  Deform_dot.row=3;   Deform_dot.col=1;
  point A_local;
  float DeltT=1.0/TT;
  float rr;

  Div=Z_Div/2;

  for (i=0; i<=TT; ++i)
    {

/****************** declare  P, P_dot, T, T_dot *********************/

       P=Pmatrix(DeltT*i);
       T=Tmatrix(DeltT*i);
       P_dot=Pdotmatrix(DeltT*i);
       T_dot=Tdotmatrix(DeltT*i);

       for (j=0; j<=Div; ++j)
          {

/****************** for ball ******************************/
            checking<<" Time:   "<<i<<"  Section:   "<<j<<endl;

            Deformation.ele[0][0]=Dispx[i][j];
            Deformation.ele[1][0]=Dispy[i][j];
            Deformation.ele[2][0]=0;

            if (Finish_YN=='Y' || Finish_YN=='y')
               {
                 Deform_dot.ele[0][0]=(0.5*Dispx_pre[i][j]-
2*Dispx[i][j]+1.5*Dispx_next[i][j])/(1*DeltT);
                 Deform_dot.ele[1][0]=(0.5*Dispy_pre[i][j]-
2*Dispy[i][j]+1.5*Dispy_next[i][j])/(1*DeltT);
                 Deform_dot.ele[2][0]=0;
               }
            else
               {
                 Deform_dot.ele[0][0]=(Dispx[i+1][j]-Dispx[i][j])/DeltT;
                 Deform_dot.ele[1][0]=(Dispy[i+1][j]-Dispy[i][j])/DeltT;
                 Deform_dot.ele[2][0]=0;
               }

            rr=sqrt(R*R-(R-R/Div*j)*(R-R/Div*j));
            Grazing_section(i,j,rr,2);
          }
```

```
        for (j=Div+1; j<=2*Div; ++j)
            {
             checking<<" Time:   "<<i<<"  Section:   "<<j<<endl;

             Deformation.ele[0][0]=Dispx[i][j];
             Deformation.ele[1][0]=Dispy[i][j];
             Deformation.ele[2][0]=0;

             if (Finish_YN=='Y' || Finish_YN=='y')
                {
                  Deform_dot.ele[0][0]=(0.5*Dispx_pre[i][j]-
2*Dispx[i][j]+1.5*Dispx_next[i][j])/(1*DeltT);
                  Deform_dot.ele[1][0]=(0.5*Dispy_pre[i][j]-
2*Dispy[i][j]+1.5*Dispy_next[i][j])/(1*DeltT);
                  Deform_dot.ele[2][0]=0;
                 }
             else
                {
                 Deform_dot.ele[0][0]=(Dispx[i+1][j]-Dispx[i][j])/DeltT;
                 Deform_dot.ele[1][0]=(Dispy[i+1][j]-Dispy[i][j])/DeltT;
                 Deform_dot.ele[2][0]=0;
                }

             rr=R;
             Grazing_section(i,j,rr,3);
            }
      }
}

/************************************************
       Output as *.ibl
   *.ibl is the file format which can be read by Pro/Engineer
   we output the grazing points section by section to enable
   Pro/E creating the solid
 *********************************/
void Output_ibl_flat(int blocknum)
{
 int i,j,flag=0;
 double deformation;
 TT=TT-1;
 for (i=0; i<=TT; ++i)
   {
     outgrazing<<"begin section ! "<<((TT+1)*blocknum)+i+1<<endl;

     outgrazing<<"begin curve ! 1 \n";
     for (j=0; j<=Z_Div; ++j)
       {
```

```
        outgrazing<<"  "<<Grazing_slice[i][j].pt[0].x<<"    "<<Grazing_slice[i][j].pt[0].y<<"
"<<Grazing_slice[i][j].pt[0].z<<endl;
        outpoint<<Grazing_slice[i][j].pt[0].x<<"    "<<Grazing_slice[i][j].pt[0].y<<"
"<<Grazing_slice[i][j].pt[0].z<<endl;
       }

     outgrazing<<"begin curve ! 2 \n";
     outgrazing<<"  "<<Grazing_slice[i][Z_Div].pt[0].x<<"
"<<Grazing_slice[i][Z_Div].pt[0].y<<"    "<<Grazing_slice[i][Z_Div].pt[0].z<<endl;
     outgrazing<<"  "<<Grazing_slice[i][Z_Div].pt[1].x<<"
"<<Grazing_slice[i][Z_Div].pt[1].y<<"    "<<Grazing_slice[i][Z_Div].pt[1].z<<endl;

     outgrazing<<"begin curve ! 3 \n";
     for (j=0; j<=Z_Div; ++j)
       {
        outgrazing<<"  "<<Grazing_slice[i][Z_Div-j].pt[1].x<<"    "<<Grazing_slice[i][Z_Div-
j].pt[1].y<<"    "<<Grazing_slice[i][Z_Div-j].pt[1].z<<endl;

        outpoint<<Grazing_slice[i][Z_Div-j].pt[1].x<<"    "<<Grazing_slice[i][Z_Div-
j].pt[1].y<<"    "<<Grazing_slice[i][Z_Div-j].pt[1].z<<endl;
       }

     outgrazing<<"begin curve ! 4 \n";
     outgrazing<<"  "<<Grazing_slice[i][0].pt[1].x<<"    "<<Grazing_slice[i][0].pt[1].y<<"
"<<Grazing_slice[i][0].pt[1].z<<endl;
     outgrazing<<"  "<<Grazing_slice[i][0].pt[0].x<<"    "<<Grazing_slice[i][0].pt[0].y<<"
"<<Grazing_slice[i][0].pt[0].z<<endl;
   }
/***************** for deformation exceed the tolerance ********/
     ofstream  point("points.pts", ios::out);
     for(i=0;i<=TT; ++i)
      {
        for(j=0;j<=Z_Div;++j)
           {
            deformation=sqrt(pow(Dispx[i][j],2)+pow(Dispy[i][j],2));
            if (fabs(deformation)>=Tolerance)
              {
                point<<Grazing_slice[i][j].pt[0].x<<"    "<<Grazing_slice[i][j].pt[0].y<<"
"<<Grazing_slice[i][j].pt[0].z<<endl;
                point<<Grazing_slice[i][j].pt[1].x<<"    "<<Grazing_slice[i][j].pt[1].y<<"
"<<Grazing_slice[i][j].pt[1].z<<endl;

              }
           }
      }

     for(i=1;i<=TT; ++i)
      {
        exceed1<<"Begin section ! "<<(TT*blocknum)+i<<endl;
```

```
        exceed1<<"Begin curve ! 1"<<endl;
        flag=0;
        float offset=0.1;
        for(j=0;j<=Z_Div;++j)
           {
            deformation=sqrt(pow(Dispx[i][j],2)+pow(Dispy[i][j],2));

            if (fabs(deformation)>=Tolerance)
              {
                exceed1<<Grazing_slice[i][j].pt[0].x+offset<<"
"<<Grazing_slice[i][j].pt[0].y+offset<<"    "<<Grazing_slice[i][j].pt[0].z<<endl;
                ++flag;
              }
           }
        if (flag==0)
          {
            exceed1<<Grazing_slice[i][0].pt[0].x<<"    "<<Grazing_slice[i][0].pt[0].y<<"
"<<Grazing_slice[i][0].pt[0].z+0.01<<endl;
            exceed1<<Grazing_slice[i][0].pt[0].x<<"    "<<Grazing_slice[i][0].pt[0].y<<"
"<<Grazing_slice[i][0].pt[0].z+0.02<<endl;
          }
        else if (flag==1)
          {exceed1<<Grazing_slice[i][0].pt[0].x<<"    "<<Grazing_slice[i][0].pt[0].y<<"
"<<Grazing_slice[i][0].pt[0].z+0.01<<endl;
          }
      }
}


/*******************************************************
  Output as point data file *.pts
*******************************************************/
void Output_point_flat()
{
  int i,j;
  int record;

/*   record=floor((LL-Depth_cut)*Z_Div/LL);    */


   for (i=0;i<=TT; ++i)
      {
        for (j=0; j<=Z_Div; ++j)
           {
            if (abs(Grazing_slice[i][j].pt[0].z)>=LL-Depth_cut)
              {
```

```
                    outpoints<<"   "<<Grazing_slice[i][j].pt[0].x<<"
"<<Grazing_slice[i][j].pt[0].y<<"     "<<Grazing_slice[i][j].pt[0].z<<endl;
                  }
              }

          for (j=0; j<=Z_Div; ++j)
             {
              if (abs(Grazing_slice[i][j].pt[1].z)>=LL-Depth_cut)
                {
                   outpoints<<"   "<<Grazing_slice[i][j].pt[1].x<<"
"<<Grazing_slice[i][j].pt[1].y<<"     "<<Grazing_slice[i][j].pt[1].z<<endl;
                  }
             }

      }
}


/*****************************************************
     output grazing sets of ballend mill
           *.ibl
*****************************************************/
void Output_ibl_ball(int blocknum)
{
 int i,j,flag=0,zero;
 double deformation;

 for (i=0; i<=TT; ++i)
   {
     outgrazing<<"begin section ! "<<((TT+1)*blocknum)+i+1<<endl;
     zero=0;
     outgrazing<<"begin curve ! 1 \n";

     for (j=0; j<=Z_Div; ++j)
       {
         if (Grazing_slice[i][Z_Div-j].pt[0].x==0 && Grazing_slice[i][Z_Div-j].pt[0].y==0 &&
Grazing_slice[i][Z_Div-j].pt[0].z==0)
            { ++zero; }
         else
            outgrazing<<"   "<<Grazing_slice[i][Z_Div-j].pt[0].x<<"
"<<Grazing_slice[i][Z_Div-j].pt[0].y<<"     "<<Grazing_slice[i][Z_Div-j].pt[0].z<<endl;
       }

     for (j=1; j<=Z_Div; ++j)
       {
         if (Grazing_slice[i][j].pt[1].x==0 && Grazing_slice[i][j].pt[1].y==0 &&
Grazing_slice[i][j].pt[1].z==0)
            { ++zero; }
         else
```

```
            outgrazing<<"  "<<Grazing_slice[i][j].pt[1].x<<"
"<<Grazing_slice[i][j].pt[1].y<<"    "<<Grazing_slice[i][j].pt[1].z<<endl;
        }

      outgrazing<<"begin curve ! 2 \n";
      outgrazing<<"  "<<Grazing_slice[i][Z_Div].pt[1].x<<"
"<<Grazing_slice[i][Z_Div].pt[1].y<<"    "<<Grazing_slice[i][Z_Div].pt[1].z<<endl;
      outgrazing<<"  "<<Grazing_slice[i][Z_Div].pt[0].x<<"
"<<Grazing_slice[i][Z_Div].pt[0].y<<"    "<<Grazing_slice[i][Z_Div].pt[0].z<<endl;

        /*    cout<<"                "<<zero<<endl;        */
    }
/***************** for deformation exceed the tolerance ********/
      ofstream  point("points.pts", ios::out);

      ofstream  exceed1("tolerance.ibl", ios::out);

      for(i=0;i<=TT; ++i)
       {
         for(j=0;j<=Z_Div;++j)
            {
             deformation=sqrt(pow(Dispx[i][j],2)+pow(Dispy[i][j],2));
             if (fabs(deformation)>=Tolerance)
               {
                 point<<Grazing_slice[i][j].pt[0].x<<"    "<<Grazing_slice[i][j].pt[0].y<<"
"<<Grazing_slice[i][j].pt[0].z<<endl;
                 point<<Grazing_slice[i][j].pt[1].x<<"    "<<Grazing_slice[i][j].pt[1].y<<"
"<<Grazing_slice[i][j].pt[1].z<<endl;

               }
            }
       }
      exceed1<<"open\n"<<"arclength\n";
      for(i=1;i<=TT; ++i)
       {
         exceed1<<"Begin section ! "<<i<<endl;
         exceed1<<"Begin curve ! 1"<<endl;
         flag=0;
         float offset=0.1;
         for(j=0;j<=Z_Div;++j)
           {
             if (Grazing_slice[i][j].pt[0].x==0 && Grazing_slice[i][j].pt[0].y==0 &&
Grazing_slice[i][j].pt[0].z==0)
                 { zero=j-1; goto jumpout;}
             else
                 { deformation=sqrt(pow(Dispx[i][j],2)+pow(Dispy[i][j],2));
                  if (fabs(deformation)>=Tolerance)
                    {
```

```
                    exceed1<<Grazing_slice[i][j].pt[0].x+offset<<"
"<<Grazing_slice[i][j].pt[0].y+offset<<"    "<<Grazing_slice[i][j].pt[0].z<<endl;
                    ++flag;
                  }
                }
              }
jumpout:   if (flag==0)
           {
             exceed1<<Grazing_slice[i][zero].pt[0].x<<"    "<<Grazing_slice[i][zero].pt[0].y<<"
"<<Grazing_slice[i][zero].pt[0].z+0.01<<endl;
             exceed1<<Grazing_slice[i][zero].pt[0].x<<"    "<<Grazing_slice[i][zero].pt[0].y<<"
"<<Grazing_slice[i][zero].pt[0].z+0.02<<endl;
           }
         else if (flag==1)
           {exceed1<<Grazing_slice[i][zero].pt[0].x<<"    "<<Grazing_slice[i][zero].pt[0].y<<"
"<<Grazing_slice[i][zero].pt[0].z+0.01<<endl;
           }
       }
}


/*************************************************************
        Calculate the Ingress at t=0;  Egress at t=1              (Not in use !!!!!)
*************************************************************/
void In_E_gress_flat(int t)
{  cout<< "Not in use !!!!";   }


/*************************************************************
     This is the main function of the program
CL data was read in block by block and generate the grazing sets block by block
All the grazing sets are output in one *.ibl file for the whole deformed swept volume
*************************************************************/
main()
{
 char tool_type;
 int  block_number;
 char unit_type;

 int  blockin;
 float Inch_to_mm=1.0;

  outgrazing<<"closed\n"<<"arclength\n";
 exceed1<<"open\n"<<"arclength\n";

 tool:   cout<<" \n"<<" Flat_end or Ball_end:   (F/B) ? ";
 cin>>tool_type;
 cin.clear();
```

```
  char  CL[10];

  cout<<" Please  input  CLdata file name:    ";
  cin>>CL;
  cin.clear();
  ifstream  input(CL, ios::in);
  if(!input)
    {
     cout<<"Can not open for read in CL data";
     exit(-1);
    }

  cout<<" How many blocks you want? :    ";
  cin>>blockin;  cout<<"        "<<blockin<<endl;
  cin.clear();

  cout<<"What's the unit of CL data: (I)nch/(M)m  ?:  ";
  cin>>unit_type;
  if (unit_type=='I' || unit_type=='i')  {Inch_to_mm=25.4;}


for (int i=0; i<=blockin; i++)
  {
    input>>XX[i]>>YY[i]>>ZZ[i]>>II[i]>>JJ[i]>>KK[i];
  }

  if (tool_type=='F' || tool_type=='f')
    {
    LL=L;

    for (block_number=0; block_number<=blockin-1; block_number++)
     {
      block=block_number;
      xx0=XX[block_number]*Inch_to_mm;
      yy0=YY[block_number]*Inch_to_mm;
      zz0=ZZ[block_number]*Inch_to_mm;
      ii0=II[block_number]; jj0=JJ[block_number]; kk0=KK[block_number];

      xx1=XX[block_number+1]*Inch_to_mm;
      yy1=YY[block_number+1]*Inch_to_mm;
      zz1=ZZ[block_number+1]*Inch_to_mm;
      ii1=II[block_number+1]; jj1=JJ[block_number+1]; kk1=KK[block_number+1];

      cout<<xx0<<"   "<<yy0<<"   "<<zz0<<"   "<<ii0<<"   "<<jj0<<"    "<<kk0<<endl;
      cout<<xx1<<"  "<<yy1<<"  "<<zz1<<"  "<<ii1<<"  "<<jj1<<"  "<<kk1<<endl;

      Deform_calculation();

      Grazingset_flat();
```

```
    Output_ibl_flat(block_number);

    Output_point_flat();

 //   In_E_gress_flat(0);    In_E_gress_flat(TT);
    }
 }

 else if (tool_type=='B' || tool_type=='b')
     {
       LL=L;

       for (block_number=0; block_number<=block; ++block_number)
        {
         xx0=XX[block_number]; yy0=YY[block_number]; zz0=ZZ[block_number];
         ii0=II[block_number]; jj0=JJ[block_number]; kk0=KK[block_number];

         xx1=XX[block_number+1]; yy1=YY[block_number+1]; zz1=ZZ[block_number+1];
         ii1=II[block_number+1]; jj1=JJ[block_number+1]; kk1=KK[block_number+1];

         Deform_calculation();

        Grazingset_ball();
        Output_ibl_ball(block_number);
        }
     }

   else
    { cout<<"Wrong selection !!"<<endl;  goto  tool;}

 outfile1.close();
 outfile2.close();
 exceed1.close();
 outgrazing.close();
 outpoints.close();
 maxdeform_out.close();
}
```

## REFERENCES


1. E.J. Armarego and N.P. Deshpande, "Computerized end milling force prediction with cutting model allowing eccentricity and cutter deflections", *CIRP Annals*, Vol.40, 1992.

2. D. Blackmore, M.C. Leu, "Applications of flows and envelopes to NC machining", *CIRP Annals*, Vol. 41, 1992.

3. D. Blackmore, M.C. Leu and Shih, F., "Analysis and modeling of deformed swept volumes", *Computer-Aided Design*, Vol. 26, 1994.

4. D. Blackmore, M.C. Leu and L. Wang, "The sweep-envelope defferential equation algorithm and its application to NC machining verification", *Computer-Aided Design*, 1996.

5. S. Boussac and A. Crosnier, "Swept volumes generated from deformable objects application to NC verification", *IEEE Robotics and Automation*, Vol. 23, 1996.

6. Hsi-Yung Feng and Chia-Hsiang Menq, "A flexible ball-end milling system model for cutting force and machining error prediction", *Journal of Manufacturing Science and Engineering*, Vol. 118, 1996.

7. H. Jiang, D. Blackmore and M.C. Leu, "The flow approach to CAD/CAM modeling of swept volumes", *Manufacturing systems: design, modeling and analysis,* Amsterdam: Elsevier, 1994

8. W.A. Kline and R.E. DeVor, "The prediction of surface accuracy in end milling", *ASME Journal of Engineering for Industry*, Vol. 104, 1982

9. K. Kops and D.T. Vo, "Determination of the equivalent diameter of an end mill based on its compliance", M.E. Report, McGill University, Canada

10. P. Lee and Y. Altintas, "Unified Mechanics and Dynamics of the Ball End Milling Process", report, Dept. of Mechanical Engineering, The University of British Columbia, Canada

11. M. C. Leu, D. Blackmore and L. Wang, "Implementation of SDE method to represent cutter swept volumes in 5-axis NC Milling", *Process International Conference on Intelligent Manufacturing*, Wuhan, China, 1995

12. M.C. Leu, L. Wang and D. Blackmore, "A verification program for 5-axis NC machining with general APT tools", *Annals of CIRP*, Vol.46, 1997

13. D. Montgomery and Y. Altintas, "Mechanism of Cutting Force and Surface Generation in Dynamic Milling", *ASME, Journal of Engineering for Industry* Vol.113, 1991

14. A. Sabberval, "Cutting force in down milling", *International Journal of Machine Tool Design Research*, Vol.23, 1983

15. S. Smith and J. Tlustry, "An overviewing of modeling and simulation of the milling process", *ASME, Journal of Engineering for Industry*, Vol.113, 1991

16. J. W. Sutherland and R.E. DeVor, "An improved method for cutting force and surface error prediction in flexible end milling system", *ASME Journal of Engineering for Industry*, Vol.108, 1986

17. J. W. Sutherland, "A dynamic model of the cutting force system in the end milling process", PhD thesis, 1987, University of Illinois at Champaign-Urbana, Illinois, USA

18. S. Takata, M.D. Tsai and M. Inui, "A cutting simulation system for machinability evaluation using a workpiece model", *Annals of CIRP*, Vol. 37, 1989

19. W. Wang and K.K.Wang, "Geometric modeling for swept volumes of moving solids", *IEEE Computer Graphics & Application*, Vol. 6, 1986

20. G. Yucesan and Y. Altintas, "Prediction of end mill forces", *ASME, Journal of Engineering for Industry*, Vol. 118, 1993